

About This Book

This reference book provides programmers with descriptions of the application programming interfaces (APIs) provided by the IBM Operating System/2 for OS/2 Warp Servers, LAN Servers, and the Directory and Security Server. These APIs provide the functional interface for server and client application development.

The OS/2 operating system referred to in this book is OS/2 Versions 2.1 through OS/2 Warp 4 (with the applicable CSDs and ServicePaks applied).

This book also provides the OS/2 Warp Connect LAN Requester APIs. These APIs are 32-bit and are generally consistent with the Net32xxx and Dos32xxx LAN APIs that ship with LAN Server 4.0.

DOS refers to DOS versions 3.3, 5.0, 6.x, or 7.0.

Who Should Use This Book

This book is written as a reference for the application or system programmer who is developing Directory and Security Server (DSS) software applications or OS/2 LAN Server software for a LAN Server, LAN Requester, OS/2 Warp Connect LAN Requester (client only), or DOS LAN Services workstation. Information in this book applies to the following servers and clients:

- LAN Server 3.0
- LAN Server 4.0
- OS/2 Warp Server
- OS/2 Warp Connect
- Directory and Security Server

This book includes DLS programming information, which is not supported by Directory and Security Server.

You should be familiar with programming the OS/2 base operating system or an equivalent multitasking operating system, and should understand the C programming language.

How This Book Is Organized

This book contains the following chapters and appendixes:

[Introduction](#), offers an overview of the LAN Server API architecture.

[Definitions and Conventions](#), provides an overview of the functions that are available to an application. This chapter contains book-wide definitions and important information to consider when programming LAN Server.

[Sample Programs](#), provides an overview of sample application resource files provided with LAN Server that you can view, print, or possibly use in your own application.

Chapters 4-37 provide a detailed reference to the LAN Server APIs, including syntax, return codes, and the data structures that are passed.

[API Return Codes](#), numerically lists the LAN Server API return codes.

[REXX RIPL Function Calls](#), provides a detailed reference to the APIs in the REXX RIPL category.

[Notes for DOS Applications](#), lists for each LAN Server API category the differences between calls from DOS LAN Service (DLS) and OS/2 workstations. It also provides a table of the header files used under DOS.

[LAN API Symbolic Constants](#), provides a list of the symbolic constants associated with variable-length ASCII strings to which data structure components used in the OS/2 LAN APIs point.

Text Highlighting Conventions

Throughout this book, the following highlighting conventions are used.

Table 1. Highlighting Conventions

HIGHLIGHTING	USED TO IDENTIFY
CAPITAL LETTERS	<ul style="list-style-type: none">o Directory nameso File nameso Acronyms
Italics	<ul style="list-style-type: none">o Book and diskette titleso Variable names and valueso Technical terms when introducedo Words of emphasis
"Monospace"	<ul style="list-style-type: none">o Coding exampleso Data structureso Function syntax templates
null	In lowercase, this term refers to the ASCII code for zero (0).
NULL	In capital letters, this term refers to the constant ((void *) 0) as defined in header file STDIO.H.
BOLD	<ul style="list-style-type: none">o Privilege levelso Field names

Double-Byte Character Set (DBCS) Information

Double-byte character set (DBCS) LAN Server runs on both single-byte character set (SBCS) systems and double-byte character set (DBCS) systems. *SBCS* is a graphic character set in which each character occupies 1 byte. *DBCS* is a graphic character set in which each character occupies 2 bytes. Languages, such as Japanese, Chinese, and Korean, that contain more symbols than can be represented by 256 code combinations require double-byte character sets.

DBCS information is included throughout this book. Restrictions for DBCS systems are indicated by the following example:

NOTE TO DOUBLE-BYTE CHARACTER SET (DBCS) USERS:

Related Books

LAN Server and Warp Server

The following books, included online with the LAN Server 4.0 product, contain information related to this book. These books will remain online even after DSS is installed:

- *IBM OS/2 LAN Server Version 4.0 Network Administrator Reference Volume 1: Planning, Installation, and Configuration*

- *IBM OS/2 LAN Server Version 4.0 Network Administrator Reference Volume 2: Performance Tuning*
- *IBM OS/2 LAN Server Version 4.0 Network Administrator Reference Volume 3: Network Administrator Tasks*

The following books, available separately, also contain information related to this book:

- *IBM Presentation Manager Programming Reference, Volume 1*
- *IBM Presentation Manager Programming Reference, Volume 2*
- *IBM Presentation Manager Programming Reference, Volume 3*
- *IBM OS/2 Technical Library, Control Program Programming Reference*
- *IBM OS/2 Technical Library, Programming Guide Volume 1*
- *IBM OS/2 Technical Library, Programming Guide Volume 2*
- *IBM OS/2 Technical Library, Programming Guide Volume 3*
- Online programming information included with your *IBM OS/2 Programming Toolkit Version 2.x*

The following books, available separately, contain important information about Multi-Protocol Transport Services and other LAN transport functions:

- *IBM Multi-Protocol Transport Services - AnyNet for OS/2: Configuration Guide*
- *IBM Multi-Protocol Transport Services - AnyNet for OS/2: Programmer's Reference*

For information about the NetBIOS protocol and the preferred NetBIOS NB30 programming interface, refer to the *IBM LAN Technical Reference*.

If you are programming DOS or 16-bit OS/2 applications and already have any of the following, you might find them to be helpful references.

- *IBM OS/2 Version 1.3 Programming Guide Technical Update*
- *IBM Presentation Manager Programming Reference Version 1.3 Technical Upgrade Volume 1*
- *IBM Presentation Manager Programming Reference Version 1.3 Technical Upgrade Volume 2*
- Online programming information included with the *IBM OS/2 Programming Toolkit Version 1.3*

You can print sections of the online books by using the Print function on the menu bar while viewing the books.

The following additional online information can be viewed from the DSS object on the desktop:

- *Online Error Message Reference*
- *README File*

The *Online Error Message Reference* and *README File* also are ASCII text files that can be viewed or printed from diskettes or CD-ROM. They are named ERROR.TXT AND README.DOC and are located in the root directory on the CD-ROM.

Directory and Security Server

The following books are included online with the Directory and Security Server product:

- *IBM Directory and Security Server Up and Running!*
- *IBM Directory and Security Server Administrator's Reference*
- *IBM Directory and Security Server Command and Utilities*
- *IBM Directory and Security Server Problem Determination*
- *IBM DCE for OS/2 Warp: Administration Command Reference*
- *IBM DCE for OS/2 Warp: Administration Guide*
- *IBM DCE for OS/2 Warp Application Development Guide - Introduction and Style Guide*
- *IBM DCE for OS/2 Warp: Application Development Guide - Core Components*
- *IBM DCE for OS/2 Warp Application Development Guide - Directory Services*
- *IBM DCE for OS/2 Warp: Application Development Reference*
- *IBM DCE for OS/2 Warp: Getting Started*
- *IBM Introduction to Distributed Computing Environment*
- *IBM DCE for OS/2 Warp Error Messages Reference*
- *IBM DCE for OS/2: MOCL Programmer's Guide and Reference*

With the exception of *Directory and Security Server Up and Running!* all books are shipped in online form. You can view the books from a CD-ROM drive, install them on your hard disk, or view them remotely (if they are installed on a server in the domain). After installation, you can access online books by selecting a book from the Directory and Security Server container.

To install or view the online books:

1. Insert the CD-ROM into the drive.
2. At an OS/2 command prompt, type *d:\INSTALL*, where *d* is the drive letter, and press Enter.
3. On the Welcome window, select **OK**.
4. Follow the instructions to install or view (CD-ROM only) the online books.

If installed, the books are installed in the DSS Books object within the Directory and Security Server container on the desktop.

Introduction

OS/2 LAN Server provides a rich set of application programming interfaces (APIs), where each API provides a particular function upon request from an application. By calling upon a combination of APIs, your application can use nearly all of the OS/2 operating system and LAN Server functions with your other applications, from file reads and writes through interprocess communication (IPC). In addition, your applications can take full advantage of true multitasking and can be integrated fully into the OS/2 desktop.

Most LAN Server API calls can be issued either locally (to an API on the same workstation where the application resides) or remotely (to another server or peer workstation). Hence, your applications can use local functions as well as LAN-wide functions.

Programmable functions include those for LAN Server itself, OS/2 LAN Requester, and DOS LAN Services (DLS). These functions are accessible to OS/2 or DLS applications, although DOS has some limitations. Any such limitations are included in either the Restrictions or DOS Considerations section of an API's description.

OS/2 LAN Server supports these 32-bit compilers:

- IBM C Set/2 C++
- Borland C++ 2.0 for OS/2
- Watcom C/C++ 10.0

Programming for Directory and Security Server (DSS) is similar to programming for LAN Server. Most LAN Server APIs are unchanged in DSS. APIs relating to security and resources have been mapped to use the services provided by Distributed Computing Environment (DCE). This allows existing applications to function unchanged on DSS and lets you build on your existing code base.

Note: Administration from LAN Server and OS/2 Warp Server servers and clients is limited.

Programming Overview

An application and LAN Server use three basic components to interact:

- API call

Issued by the application, an *API call* tells LAN Server or LAN Requester which function to invoke.

- Return code

Issued by LAN Server or LAN Requester, the *return code* tells the application whether the function was successful or, if not, why it failed.

- Data structures

When an API call is issued, pertinent data (for example, a user ID or workstation profile) also is passed between the application and LAN Server or LAN Requester. This data is placed in a *data structure* in local workstation memory, where both LAN Server and your application can read and write to it.

Depending on the nature of the API, this data can be sent:

- From the caller to the requester
- From the caller to the server
- From the server to the caller
- From the requester to the caller

Any call issued to an API must contain pointers to the associated data structure whether the application is sending or requesting data.

Each type of API has one or more unique data structures associated with it. Typically, the *but*, *usButlen*, and *sLevel* parameters (described in [Common Parameter Definitions](#)) are used with an API call to specify which data structure to use.

Structure of a Typical API Call

A typical API call contains the following:

- A keyword that addresses the call either to LAN Server or to an OS/2 function. This keyword usually is one of the following:

<i>Net</i>	Invokes a LAN Server function.
<i>Dos</i>	Invokes an OS/2 operating system function (except for the Mailslot Category).
<i>Spl</i>	Invokes spooler functions.
<i>UPM</i>	Invokes a User Profile Management (UPM) function.
- A keyword that identifies the function category.

For example, *User* identifies a function that controls user accounts, and *Alert* identifies a message alert function. The category identifier can be compound, such as *ServerDisk* (specifies a function on the server disk) or *PrintQ* (specifies a printer queue function).
- A keyword that describes the action to be performed.

Examples include *Add*, *Del*, *Read*, and *Purge*.
- An optional keyword that sometimes is used to identify the object upon which the action is to be performed.

For example, NetPrintQPurgeSelf deletes only the queued print jobs submitted by the application requesting the purge, whereas NetPrintQPurge deletes all queued print jobs from the specified queue.
- One or more parameters in parentheses () that provide specific information about the operation to be performed.

Parameters specify which data structure to use (if there is one) and provide other details applicable to a particular API. Some parameters that are used commonly with many APIs are defined in [Definitions and Conventions](#). Exceptions or unique parameters are defined where they occur in this book.

Return Codes and Symbolic Constants

For each API call issued, LAN Server returns a numeric code telling the application whether the function was successful or why it failed. (Unless noted otherwise in the Syntax sections, the return function type is *API_FUNCTION*.) To make programming easier, each of these codes also is assigned short, descriptive text, called a *symbolic constant* in this book. For example, the symbolic constant *NETERR_Success* is equivalent to a return code of 0, a successful completion. (A symbolic constant is the same as a *manifest* or *manifest constant*, terms used in previous releases.)

The text for the symbolic constant is defined in the header files (also called *include* files in previous releases). The header files provided by LAN Server are installed in the \IBMLAN\NETSRC\H subdirectory by default, although this can be changed.

Your application must include any header files that contain symbolic constants used by your application. Although an application typically uses more than one header file, each header file needs to be included only once, usually at the beginning of the program.

Data Structures

Data structures are used to pass information between an application and the LAN Server APIs. Most APIs use data structures, and many data structures have more than one level of information available. Level 0 (zero) specifies the least information, and higher levels usually specify incrementally more information.

For instance, if you issue a `NetServerGetInfo` call at level 0, only the ASCII name of the server is returned. If you issue the same call at level 3, the returned data structure will contain 47 entries: the name of the server plus all level 1 and level 2 data, including the number of users allowed on the server, server heuristics, and so on.

DSS API Strategy

DSS is a combination of technologies which includes many tools and APIs for distributed application development, administration and system management. The DSS API strategy guides the application developer by helping to determine which API should be used when multiple APIs could perform the desired function.

When the activity is porting, either from existing LAN Server applications to DSS applications or from existing application to family applications, one of the major objectives is to minimize the amount of rewrite. To that end, and to the extent that is possible, developers should use existing LAN Server APIs when porting applications. For the most part this will work for everything except the manipulation of ACLs on DSS servers.

When family application are ported or newly written, the strategy should be similar to that for porting applications in general. Family applications must determine if the target of an operation is a LAN Server, Warp Server, or DSS workstation before issuing API calls that only work in one environment or the other.

Scoping Rules

In DSS, you can have multiple resource domains within a cell and this will most likely be typical of your network's topography. In order to maintain compatibility with your existing applications and still provide access to the full range of tools available in DSS, several API categories introduced scoping rules. These APIs are:

- `NetUser`
- `NetGroup`
- `NetApp`
- `NetAlias`

The first parameter (*pszServername*) in LAN Server 4.0 took the name of a server to execute on. In DSS, the use of this parameter has been expanded and is now called *pszTarget*. You can still pass in a servername, but the API scopes the call to the server's resource domain.

The following examples help introduce scoping rules.

We will start with a cell by the name of `/.../mybigcell` with two resource domains, `resdom_1` and `resdom_2`. The domain `resdom_1` has `SERVER1` while `resdom_2` has `SERVER2`.

Example 1: Scoped to a server's resource domain.

```
NetUserEnum ("\\SERVER1", ... )
This returns all users belonging to resdom_1.

NetUserEnum ("\\SERVER2", ... )
This returns all users belonging to resdom_2.
```

Example 2: Scoped to a cell.

```
NetUserEnum ("/ ... /anycell", ...)
This returns all DSS users in / ... /anycell.

NetUserEnum ("/.:", ... )
This returns all DSS users in the local cell.
```

Example 3: Scoped to a resource domain.

```
NetUserEnum ("//resdom_1", ... )
```

This returns all the users associated with resource domain resdom_1.

```
NetUserEnum ("//bldg901@/ ... /devcell", ... )
This returns all the users in cell / ... /devcell associated
with resource domain //bldg901.
```

OTHGRPS/OTHALIAS/OTHUSERS

These special entries provide a way for enumeration APIs to indicate that information outside of the scope is available. For example, NetGroupGetUsers returns all of the users in resdom 1 that are members of GRP1 when (*//resdom_1, ...*) is used and GRP1 is represented as It is possible there could be other users in the group which are not actually part of the resource domain. If this is the case, the API returns OTHUSERS. This informs you that there are others in the group that are members of another resource domain.

The following table illustrates a sample DSS cell containing two resource domains: deptJRAS and Development.

Table 2. Sample DSS Cell (/ ... /samplecell)

RESOURCE DOMAIN: DEPTJRAS	RESOURCE DOMAIN: DEVELOPMENT
Users:	
JANE - Administrator	JANE - User
JOE - User, print-operator	JACK - Administrator
JACK - User	SUSAN - User
Groups:	
o GRP1	o CODERS
- JANE	- JANE
- JACK	- JOE
o GRP2	- dceuser
	o TESTERS
	o GRP2
Servers:	
S1	DEVSrv
Alias: UTILITIES	Alias: COMPILER
Application: LOTUS 1-2-3	Application: IPMD
Non-DSS Users	
o Non-DSS Users	
- cell_admin	
Non-DSS Groups	
o laser_users	
- cell_admin	
- JANE	

Here are several examples of scoping rules and the return of OTHGRPS and OTHUSERS. These actions also apply to OTHALIAS.

Examples:

- 1) Enumerate the users in resource domain deptJRAS.
NetUserEnum("//deptJRAS", ...) or NetUserEnum("\\\\S1", ...)
API will return JANE, JOE, and JACK.
- 2) Enumerate the users in resource domain Development.
NetUserEnum("//Development", ...) or NetUserEnum("\\\\DEVSRV", ...)
API will return JANE, JACK, and SUSAN.
- 3) Enumerate the users in the local cell.
NetUserEnum("/ ... /samplecell", ...) or NetUserEnum("/.:", ...)
API will return JANE, JOE, JACK, SUSAN.
- 4) Enumerate the groups in the cell in which JOE is a member.
NetUserGetGroups("/ ... /samplecell", "JOE", ...)
API will return CODERS.
- 5) Enumerate the groups in resource domain deptJRAS in which JOE is a member.
NetUserGetGroups("//deptJRAS", "JOE", ...) or
NetUserGetGroups("\\\\S1", "JOE", ...)
This API returns, OTHGRPS. JOE is not a member of any groups
in resource domain deptJRAS. OTHGRPS is returned to indicate
that JOE is a member of groups outside of the resource domain.
- 6) Enumerate the groups in the local cell in which JANE is a member.
NetUserGetGroups("/.:", "JANE", ...) or
NetUserGetGroups("/ ... /samplecell", "JANE", ...)
This API returns GRP1, CODERS and OTHGRPS. OTHGRPS is returned
because JANE is a member of laser-users, which is not a valid
name for a DSS group.

DSS provides expanded capabilities with the existing Net APIs by expanding the valid values for *pszTarget*. However, if an application must run on non-DSS clients, you should either not use the new forms or determine what system is running the application and dynamically use the correct value.

DCE and the Managed Object Class Library

To access the full range of services provided by DSS, you may need to use DCE remote procedure calls (RPCs) or MOCL. DCE provides POSIX Threads (pthreads), Remote Procedure Calls (RPCs), Cell Directory Service (CDS), Security Service and Time Service.

The MOCL provides a client-side object-oriented interface for managing DCE services. For more information on programming to DCE and MOCL, see:

- *IBM DCE for OS/2 Warp Application Development Guide - Introduction and Style Guide*
- *IBM DCE for OS/2 Warp: Application Development Guide - Core Components*
- *IBM DCE for OS/2 Warp Application Development Guide - Directory Services*
- *IBM DCE for OS/2 Warp: Application Development Reference*
- *IBM DCE for OS/2: MOCL Programmer's Guide and Reference*

Definitions and Conventions

This chapter provides definitions and other important information to consider when writing an application to the LAN Requester Version 4.0 API conventions.

16-Bit and 32-Bit Support

LAN Server, Warp Server and DSS provide support for both 16-bit and 32-bit applications. Support for both types of applications allows a gradual change of LAN Server APIs into 32-bit worker routines without requiring 32-bit applications to change.

The new APIs were created with pure 32-bit programming in mind. In most cases, however, the 32-bit API is an entry point to 16-bit code. This implies that you must use a compiler, such as IBM C Set/2, with these characteristics:

- It can cause the addresses in the process's Local Descriptor Table to be tiled, allowing simple conversion from flat addressing to *selector:offset* and back.
- It allocates memory objects (such as data buffers) to prevent them from crossing 64KB memory boundaries.

The second characteristic has a further implication. Just as in the 16-bit world, buffer sizes are limited to 64KB in length. No matter how large a buffer you supply, an API rarely returns more than 64KB of data at a time. The exceptions are Net32AliasEnum and Net32UserEnum (OS/2 only), which can return more than 64KB of data.

Note: In DSS, all Net32User and Net32Group APIs support buffers larger than 64KB.

A 32-bit application that uses the Net32 or the Dos32 APIs must define the constant PURE_32 (not INCL_32) before including any LAN Server header files. The constant INCL_32 is for mixed-model programs only. The 32-bit API interfaces are slightly different from the 16-bit interfaces. The main highlights of the new APIs are:

- Parameters
 - The number of parameters is the same for both versions, except that a *pszReservedX* parameter has been added to the 32-bit APIs in the following categories:

CATEGORY	APIS USING pszReservedX
Alias	Net32AliasXXXX
Application	Net32AppXXXX
Group	Net32GroupXXXX
User	Net32UserXXXX

- All variables declared *char* in LAN Server Version 3.0 are now *unsigned char* for Version 4.0.
- All Version 3.0 pointers to *char* are now pointers to *unsigned char*.
- Note:** Applications must be compiled with the compiler option that defaults *char* to *unsigned char*.
- Most (but not all) *short* integers, whether signed or unsigned, have become *unsigned long* integers in the 32-bit version. This includes reserved parameters.
- All pointers to *short* integers, signed or unsigned, have become pointers to *unsigned long* integers in the 32-bit version. This includes reserved parameters.

- Data structures

All data structures and their alignment are unchanged (except that *char* becomes *unsigned char*). Some 4-byte elements might be aligned on 16-bit WORD boundaries rather than 32-bit DWORD boundaries. This is a compromise between the speed of execution of the API code and application code.

- Names

- The names of the Mailslot APIs are changed from 16-bit DosXXX to 32-bit Dos32XXX. The 16-bit NetXXX APIs are changed to Net32XXX for the 32-bit names. The NetBiosXXX APIs are named NetBios32XXX for 32-bit, and the UPMXXX APIs are U32XXX for 32-bit applications.
- The names of the data structures are the same for all. This means that for a given data structure, you cannot not mix pure 16-bit and 32-bit code. You can, however, write programs that use the mixed model (INCL_32 defined) in one file and pure 32-bit code (PURE_32 defined for OS/2) in another. You must choose only one type (pure 16-bit, mixed,

or pure 32-bit) for each program file.

- Return Codes

The Net32XXX APIs return 16-bit return codes.

Note: Not all 16-bit APIs have 32-bit entry points. For example, because 16- and 32-bit file handles are incompatible, there are no Net32Handle APIs. In addition, all the APIs in the following categories have no thunked counterparts:

Named Pipe
Print Destination
Print Job
Print Queue
Spooler

API Security

OS/2 LAN Server access security is used to assign, verify, and enforce access to resources for each user or group ID. (Access privileges must be granted by a user with administrative authority.)

Many APIs have access restrictions and require administrative authority to be invoked. These types of APIs usually retrieve or set sensitive data or control key network services. Specific access requirements are specified under the Restrictions section of each API described in this book. LAN Server provides the following types of access protection at the API level:

- Administrative privilege level, which includes full access
- Various access privilege levels for remote API function calls
- Various access privilege levels for local API function calls if Local Security is installed

Most of the APIs can be run on local or remote workstations, provided that the required software services are running. Access restrictions are not enforced for local calls to LAN Server APIs unless Local Security is installed on the server.

The LAN Server access control subsystem (ACS) controls the domain-wide access of resources by users and groups. LAN Server maintains users, groups, and resource access controls in a user accounts subsystem (UAS) database.

Remote Protection

All calls to remote server APIs are subject to access privilege checking. A user must be logged onto the domain in order to access any remote LAN Server API functions, utilities, or other resources.

Users with administrator authority have maximum access to the functions provided by LAN Server APIs. Users without administrator authority must be granted permission to access shared resources by an administrator. See also the discussions about access authority in the introductions to [Group Category](#) and [User Category](#).

DSS

In DSS, security is enforced by ACLs for NetAccess, NetAlias, NetApp, NetUser and NetGroup APIs rather than privilege level checking.

Local Security

If Local Security is installed on a LAN Server workstation, the files on the 386-HPFS drive and all APIs on that server are protected against

unauthorized access by a user or application at that server. Access to files on a drive with Local Security is similar to a redirected drive (except that a local user can read or run local files unless restricted by an administrator).

All APIs on a server with Local Security, however, require the same administrative authority as calls to a remote server. LAN Server verifies that the caller has sufficient access privilege for the requested function. Furthermore, Local Security can be invoked at the API level even if it is not installed on a LAN Server drive.

Naming Conventions

Naming conventions for the ASCII strings used with LAN Server APIs depend on the type of name. For instance, server or resource names follow different rules than the names for users or groups.

A server is named during the installation of the LAN Server code, where the name is placed in the IBMLAN.INI file in the *computename* parameter. Subsequently, an application always must refer to that server by that name, preceded by a double backslash (\\).

Any shared resource can be named according to the universal naming convention (UNC). A UNC name consists of a double backslash, the name of the server, another backslash, and the name of the resource: *\\servername\resource*.

Any shared resource (or portion of a UNC name) can be assigned an alias name of up to 8 bytes. Once assigned an alias, a resource can be addressed simply by the name of the alias, regardless of the server to which the resource is connected.

On HPFS partitions, LAN Server supports directory names of up to 260 bytes, including:

- The drive letter
- A colon (:)
- All characters in the directory path, including backslashes (\) and slashes (/)
- The file or directory name on the end of the path
- The null character at the end of the ASCII string

For file allocation table (FAT) disk partitions and all DLS requesters, file names can be up to 8 bytes long; extensions, up to 3 bytes; individual path names, up to 63 bytes; and a fully designated path name, up to 128 bytes.

Other naming restrictions for OS/2 requesters and servers include the following:

- User or group IDs can contain up to 20 bytes, not including the null character at the end of the string.
- Passwords can be up to 14 bytes, not counting the null character at the end.
- Requester, server, and domain names can be up to 15 bytes each, not counting the null character at the end.
- User and group IDs, passwords, workstation names, and domain names cannot use the following characters:
 - ASCII characters less than hexadecimal 21
 - Any of the following:
" / \ [] : | < > + =
- Periods (.) can be used; however, a period cannot be used as the first character in a network name, nor can 2 periods be used adjacently. For instance, .work.sta.1 and work..sta.1 are not valid, because of the placement of the periods. The name work.sta.1 is valid.

[LAN API Symbolic Constants](#) and the NETCONS.H header file provide additional information about the formats and lengths of symbolic constants used by LAN Server. For more information about UNC naming conventions and aliases, see the *IBM OS/2 LAN Server Version 4.0 Network Administrator Reference Volume 1: Planning, Installation, and Configuration*.

Common Parameter Definitions

This section lists the parameters commonly used in this book. Exceptions are described where they occur. Unless stated otherwise, all ASCII strings mentioned in the following list conform to the naming conventions in [Naming Conventions](#).

Tip: Copy this section (or print it if you are viewing the online version) and the table of contents or index to use alongside the

APIs described later in this book.

<i>buf</i>	(unsigned char LSFAR *) points to the local buffer address of the data structure to be sent or received with an API call.
<i>pStatusbuf</i>	(unsigned char LSFAR *) points to a buffer of size <code>lsdce_err_stat_t</code> (defined in <code>lsdceerr.h</code>). The <i>pStatusbuf</i> contains specific DCE error information when <code>NERR_DCEError</code> is returned. It must be NULL for LAN Server and Warp Server requests.
<i>pszDirPath</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of the directory upon which the API is to act. The ASCIIZ name is in the form of a complete directory path, beginning with the drive letter and ending without a backslash (for example, <code>C:\BMLAN\LOGS</code>). The only exception is a root directory, which ends with a backslash (for example, <code>C:\</code>).
<i>pszGroupID</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the group ID upon which the API is to act.
<i>pszQueueName</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of a particular printer queue.
<i>pszReservedX</i>	(unsigned char LSFAR *) is an extra parameter. Presently, it is reserved and must be NULL.
<i>pszResource</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of the local resource:

RESOURCE TYPE	ASCIIZ STRING FORMAT
Drive	<code>driveletter:</code>
Directory	<code>driveletter:\directorypath</code>
File	<code>driveletter:\directorypath\</code>
Named Pipe	<code>\PIPE\pipename</code>
Spooler queue	<code>\PRINT\queueName</code>
Serial device queue	<code>\COMM\chardevqueue</code>

pszServername (const unsigned char LSFAR *) points to an ASCIIZ string containing the network name of the server that is to run the API call. Server names begin with a double backslash (`\\servername`). A NULL pointer or string indicates that the API is to be invoked on the local computer.

pszUserID (unsigned char LSFAR *) points to an ASCIIZ string containing a particular user ID.

pusBytesAvail or *pulBytesAvail* (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the number of bytes currently available from an API. (Sometimes this parameter is used with reiterative calls when *pusBytesReturned* does not include all of the available data; in this case, you can use *pusBytesAvail* to determine the next *usBuflen* size or the number of reiterations required to return all of the available data.)

pusBytesReturned or *pulBytesReturned* (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the number of bytes returned from an API to the *buf* data structure. (Since this is the actual size of the returned data structure, this number always is less than or equal to *usBuflen*.)

pusEntriesAvail or *pulEntriesAvail* (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the number of data structures currently available from an API. (Sometimes this parameter is used with reiterative calls when *pusEntriesReturned* does not include all of the available entries; in that case, you can use *pusEntriesAvail* to determine the next *usBuflen* size or the number of reiterations required to return all of the available data.)

pusEntriesReturned or *pulEntriesReturned* (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the number of data structures returned from an API. (For example, in retrieving the same information about all users logged on to a particular server, a sequential list of data structures can be returned, one for each user.)

sIncludeSubdirs (16-bit short or 32-bit unsigned long) sometimes is used with the *pszDirPath* parameter for `XXXEnum` and `XXXCheck` APIs to specify whether the API is to act upon all of the subdirectories under the *pszDirPath* tree. If the value of this parameter is 0, only the specified *pszDirPath* is acted upon. If the value is nonzero, all subdirectories under *pszDirPath* are included.

<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) specifies which level of a data structure to use, where each level contains a unique set of information fields. (The levels of a data structure often are arranged in a hierarchy in which each level contains the data of its preceding level plus additional data fields.)
<i>usBuflen</i> or <i>ulBuflen</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the amount of local memory (in bytes) allocated to the <i>buf</i> data structure.
<i>usJobId</i>	(unsigned short) specifies the identification number for a particular print job.

Data Structures and Buffer Sizes

For both 16-bit and 32-bit applications, each buffer length (specified by *usBuflen* or *ulBuflen*) is limited to 64KB. The exceptions are Net32AliasEnum and Net32UserEnum (OS/2 only), which can return greater than 64KB. For DSS only, all Net32User and Net32Group APIs support buffers larger than 64KB.

When a data structure contains one or more pointers to variable-length data (such as an ASCIIZ string) and is passed between an application and an API, that application must provide a buffer large enough to store both the fixed-length and variable-length data. (Buffer size is specified by the 16-bit *usBuflen* or 32-bit *ulBuflen* parameters.) Otherwise, not all of the data can be passed or returned. If the buffer is too small for the fixed-length data, the API returns NERR_BufTooSmall.

If a buffer is too small to hold all variable-length data associated with a structure, your application should notify the API that no variable-length data is being passed. This is done by specifying NULL pointers to the variable-length data.

If an application calls an API that could return more variable-length data than the buffer can store, that API returns as much data as possible, setting any pointers to data not returned to NULL. In this case, the API also returns the code ERROR_MORE_DATA (value 234) and the number of bytes required to store all available data (in the *pusBytesAvail* or *pulBytesAvail* value). For some APIs, you can get all the data you want by repeatedly calling the API, such as NetFileEnum2 (or Net32FileEnum2). LAN Server provides a sample program (FILEEN32.C) demonstrating this feature of Net32FileEnum2. See [Sample Programs](#) for an overview of the sample programs provided with LAN Server.

For XXXGetInfo APIs, you can determine the necessary buffer size by calling the GetInfo API with the level parameter set to the level of data structure you want and *usBuflen* (or *ulBuflen*) set to 0. In this case, the GetInfo API returns the number of bytes available from that API in the *pusBytesAvail* (or *pulBytesAvail*) value. Then you can call the same API again and allocate a buffer size (*usBuflen* or *ulBuflen*) at least as large as the returned bytes available.

Note: This technique should not be used on DSS for NetUser, NetGroup, NetApp, or NetAlias APIs because it may affect the performance of your application.

Exception	When an application passes a data structure to the XXXAddInfo or XXXSetInfo APIs, the size of the buffer needs to be only as large as the fixed-length portion of the data structure. You do not need to include space for the variable-length data (if any).
-----------	---

The fixed-length and variable-length data do not have to be contiguous in the same memory region, even for remote calls.

Linking an Application to OS/2 Warp LAN Requester Function Libraries

This section provides a description of the LAN Server function libraries and an alphabetical list of the LAN Server APIs that require each library.

Note: The function names are spelled in the uppercase and lowercase style required for C language programs. Information is OS/2-specific.

Programs link to standard libraries (.LIBs) and dynamic link libraries (.DLLs). Standard libraries provide information such as the name of a dynamically linked run-time library to the relocatable object code at link time. Dynamic link libraries contain the actual assembler code of a function and are run at run time.

Link-Time Libraries

At link time, any program that calls a particular API function must be linked to a library (.LIB) containing information about the function. Link-time libraries provide information that allows the operating system dynamically to link the appropriate dynamic link library (.DLL) to a program at program load time. The default directory where each API is installed is given.

Note: You can use dynamic loading and address resolution instead of the import libraries specified here. See the *IBM OS/2 Programming Toolkit* online programming information or the *OS/2 Technical Library, Programming Guides* for more information about these techniques.

If calling any of the following APIs, a program must be linked to the NETAPI.LIB library (stored in \IBMLAN\NETSRC\OS2\LIB):

NetAccessAdd
NetAccessApply
NetAccessCheck
NetAccessDel
NetAccessEnum
NetAccessGetInfo
NetAccessGetUserPerms
NetAccessSetInfo
NetAlertRaise
NetAlertStart
NetAlertStop
NetAliasAdd
NetAliasDel
NetAliasEnum
NetAliasGetInfo
NetAliasSetInfo
NetAppAdd
NetAppDel
NetAppEnum
NetAppGetInfo
NetAppSetInfo
NetAuditClear
NetAuditRead
NetAuditWrite
NetBiosClose
NetBiosEnum
NetBiosGetInfo
NetBiosOpen
NetBiosSubmit
NetCharDevControl
NetCharDevEnum
NetCharDevGetInfo
NetCharDevQEnum
NetCharDevQGetInfo
NetCharDevQPurge
NetCharDevQPurgeSelf
NetCharDevQSetInfo
NetConfigGet2
NetConfigGetAll2
NetConnectionEnum
NetCreateRIPLMachine
NetDASDAdd
NetDASDCheck
NetDASDCtl
NetDASDDel
NetDASDEnum
NetDASDGetInfo
NetDASDSetInfo
NetDeleteRIPLMachine
NetErrorLogClear
NetErrorLogRead
NetErrorLogWrite
NetFileClose2
NetFileEnum2
NetFileGetInfo2
NetGetDCName
NetGetRIPLMachine
NetGroupAdd
NetGroupAddUser
NetGroupDel
NetGroupDelUser
NetGroupEnum
NetGroupGetInfo
NetGroupGetUsers
NetGroupSetInfo

NetGroupSetUsers
 NetHandleGetInfo
 NetHandleSetInfo
 NetLogonEnum
 NetMessageBufferSend
 NetMessageFileSend
 NetMessageLogFileGet
 NetMessageLogFileSet
 NetMessageNameAdd
 NetMessageNameDel
 NetMessageNameEnum
 NetMessageNameFwd
 NetMessageNameGetInfo
 NetMessageNameUnFwd
 NetRemoteCopy
 NetRemoteExec
 NetRemoteMove
 NetRemoteTOD
 NetServerAdminCommand
 NetServerDiskEnum
 NetServerEnum2
 NetServerGetInfo
 NetServerSetInfo
 NetServiceControl
 NetServiceEnum
 NetServiceGetInfo
 NetServiceInstall
 NetServiceStatus
 NetSessionDel
 NetSessionEnum
 NetSessionGetInfo
 NetSetRIPLMachine
 NetShareAdd
 NetShareCheck
 NetShareDel
 NetShareEnum
 NetShareGetInfo
 NetShareSetInfo
 NetStatisticsGet2
 NetUseAdd
 NetUseDel
 NetUseEnum
 NetUseGetInfo
 NetUserAdd
 NetUserDCDBInit
 NetUserDel
 NetUserEnum
 NetUserGetAppSel
 NetUserGetGroups
 NetUserGetInfo
 NetUserGetLogonAsn
 NetUserModalsGet
 NetUserModalsSet
 NetUserPasswordSet
 NetUserSetAppSel
 NetUserSetGroups
 NetUserSetInfo
 NetUserSetLogonAsn
 NetUserValidate2
 NetWkstaGetInfo
 NetWkstaSetInfo
 NetWkstaSetUID2

If calling any of the following APIs, a program must be linked to the NETAPI32.LIB library (stored in \IBMLAN\NETSRC\OS2\LIB):

Net32AccessAdd
 Net32AccessApply
 Net32AccessCheck
 Net32AccessDel
 Net32AccessEnum
 Net32AccessGetInfo
 Net32AccessGetUserPerms
 Net32AccessSetInfo
 Net32AlertRaise
 Net32AlertStart
 Net32AlertStop
 Net32AliasAdd

Net32AliasDel
Net32AliasEnum
Net32AliasGetInfo
Net32AliasSetInfo
Net32AppAdd
Net32AppDel
Net32AppEnum
Net32AppGetInfo
Net32AppSetInfo
Net32AuditClear
Net32AuditRead
Net32AuditWrite
Net32CharDevControl
Net32CharDevEnum
Net32CharDevGetInfo
Net32CharDevQEnum
Net32CharDevQGetInfo
Net32CharDevQPurge
Net32CharDevQPurgeSelf
Net32CharDevQSetInfo
Net32ConfigGet2
Net32ConfigGetAll2
Net32ConnectionEnum
Net32CreateRIPLMachine
Net32DASDAdd
Net32DASDCheck
Net32DASDCtl
Net32DASDDel
Net32DASDEnum
Net32DASDGetInfo
Net32DASDSetInfo
Net32DelRIPLMachine
Net32EnumRIPLMachine
Net32ErrorLogClear
Net32ErrorLogRead
Net32ErrorLogWrite
Net32FileClose2
Net32FileEnum2
Net32FileGetInfo2
Net32GetDCName
Net32GetRIPLMachineInfo
Net32GroupAdd
Net32GroupAddUser
Net32GroupDel
Net32GroupDelUser
Net32GroupEnum
Net32GroupGetInfo
Net32GroupGetUsers
Net32GroupSetInfo
Net32GroupSetUsers
Net32LogonEnum
Net32MessageBufferSend
Net32MessageFileSend
Net32MessageLogFileGet
Net32MessageLogFileSet
Net32MessageNameAdd
Net32MessageNameDel
Net32MessageNameEnum
Net32MessageNameFwd
Net32MessageNameGetInfo
Net32MessageNameUnFwd
Net32RemoteCopy
Net32RemoteExec
Net32RemoteMove
Net32RemoteTOD
Net32ServerAdminCommand
Net32ServerDiskEnum
Net32ServerEnum2
Net32ServerGetInfo
Net32ServerSetInfo
Net32ServiceControl
Net32ServiceEnum
Net32ServiceGetInfo
Net32ServiceInstall
Net32ServiceStatus
Net32SessionDel
Net32SessionEnum
Net32SessionGetInfo
Net32SetRIPLMachineInfo


```

Net32ShareAdd
Net32ShareCheck
Net32ShareDel
Net32ShareEnum
Net32ShareGetInfo
Net32ShareSetInfo
Net32StatisticsGet2
Net32UseAdd
Net32UseDel
Net32UseEnum
Net32UseGetInfo
Net32UserAdd
Net32UserDCDBInit
Net32UserDel
Net32UserEnum
Net32UserGetAppSel
Net32UserGetGroups
Net32UserGetInfo
Net32UserGetLogonAsn
Net32UserModalsGet
Net32UserModalsSet
Net32UserPasswordSet
Net32UserSetAppSel
Net32UserSetGroups
Net32UserSetInfo
Net32UserSetLogonAsn
Net32UserValidate2
Net32WkstaGetInfo
Net32WkstaSetInfo
NetBios32Close
NetBios32Enum
NetBios32GetInfo
NetBios32Open
NetBios32Submit

```

If calling any of the following APIs, a program must be linked to the HPFS386.LIB library (stored in \IBMLAN\NETSRC\OS2\LIB):

```

HPFS386GetInfo
HPFS386GetInfo16

```

If calling any of the following APIs, a program must be linked to either the OS2286.LIB library (for 16-bit applications) or the OS2386.LIB library (for 32-bit applications):

```

DosBufReset
or 32-bit DosResetBuffer
DosCallNmPipe
or 32-bit DosCallNPipe
DosClose
DosConnectNmPipe
or 32-bit DosConnectNPipe
DosDisconnectNmPipe
or 32-bit DosDisconnectNPipe
DosMakeNmPipe
or 32-bit DosCreateNPipe
DosOpen
DosPeekNmPipe
or 32-bit DosPeekNPipe
DosPrintDestAdd
or 32-bit SplCreateDevice
DosPrintDestControl
or 32-bit SplControlDevice
DosPrintDestDel
or 32-bit SplDeleteDevice
DosPrintDestEnum
or 32-bit SplEnumDevice
DosPrintDestGetInfo
or 32-bit SplQueryDevice
DosPrintDestSetInfo
or 32-bit SplSetDevice
DosPrintJobContinue
or 32-bit SplReleaseJob
DosPrintJobDel
or 32-bit SplDeleteJob
DosPrintJobEnum
or 32-bit SplEnumJob
DosPrintJobGetId

```

DosPrintJobGetInfo
 or 32-bit SplQueryJob
 DosPrintJobPause
 or 32-bit SplHoldJob
 DosPrintJobSetInfo
 or 32-bit SplSetJob
 DosPrintQAdd
 or 32-bit SplCreateQueue
 DosPrintQContinue
 or 32-bit SplReleaseQueue
 DosPrintQDel
 or 32-bit SplDeleteQueue
 DosPrintQEnum
 or 32-bit SplEnumQueue
 DosPrintQGetInfo
 or 32-bit SplQueryQueue
 DosPrintQPause
 or 32-bit SplHoldQueue
 DosPrintQPurge
 or 32-bit SplPurgeQueue
 DosPrintQSetInfo
 or 32-bit SplSetQueue
 DosQFHandState
 or 32-bit DosQueryFHState
 DosQHAndState
 or 32-bit DosQueryHType
 DosQNmPHAndState
 or 32-bit DosQueryNPHState
 DosQNmPipeInfo
 or 32-bit DosQueryNPipeInfo
 DosQNmPipeSemState
 or 32-bit DosQueryNPipeSemState
 DosRead
 DosReadAsync
 DosSetFHandState
 or 32-bit DosSetFHState
 DosSetNmPHAndState
 or 32-bit DosSetNPHState
 DosSetNmPipeSem
 or 32-bit DosSetNPipeSem
 DosTransactNmPipe
 or 32-bit DosTransactNPipe
 DosWaitNmPipe
 or 32-bit DosWaitNPipe
 DosWrite
 DosWriteAsync
 SplQmAbort
 SplQmClose
 SplQmEndDoc
 SplQmOpen
 SplQmStartDoc
 SplQmWrite

If calling any of the following APIs, a program must be linked to the MAILSLIB.LIB library (stored in \BMLAN\NETSRC\OS2\LIB):

DosDeleteMailslot
 DosMailslotInfo
 DosMakeMailslot
 DosPeekMailslot
 DosReadMailslot
 DosWriteMailslot
 Dos32DeleteMailslot
 Dos32MailslotInfo
 Dos32MakeMailslot
 Dos32PeekMailslot
 Dos32ReadMailslot
 Dos32WriteMailslot

If calling any of the following APIs, a program must be linked to the UPM.LIB library (stored in \BMLAN\NETSRC\OS2\LIB by default):

UPMELOCL
 UPMELOCU
 UPMEULGF
 UPMEULGN
 UPMEUSRL
 U32ELOCL

U32ELOCU
U32EULGF
U32EULGN
U32EUSRL

If you are writing an application that is to run on LAN Server Version 2.0 or 3.0, you must link your application to NETOEM.LIB as well as the NETAPI.LIB library (or NETAPI32.LIB) library. Link to the NETOEM.LIB library before the other library files; for example:

```
link myprogram, ,netoem.lib+netapi.lib+os2.lib;
```

LAN Server installs the NETOEM.LIB library in the same directory as the other library files \IBMLAN\NETSRC\OS2\LIB by default.

Run-Time Libraries

At run time, any program that calls a particular API function must be linked dynamically to a library containing the binary executable files for that function. The OS/2 operating system automatically links the program and library together when a particular function is called.

LAN Server provides the following dynamic link libraries (DLLs):

LIBRARY	CONTENTS	DIRECTORY
MAILSLOT.DLL	Mailslot API library	\MUGLIB\DLL
NETAPI.DLL	Base network API library	\MUGLIB\DLL
NETAPI32.DLL	32-bit base network API library	\MUGLIB\DLL
NETOEM.DLL	Stub library for legacy applications	\MUGLIB\DLL

API Services Requirements and Operator Privileges

This section provides a table that lists each API, its software services and administrative requirements, and whether a local-only library is available for that API. This table also lists the API access privileges allowed to a user with a *server operator* level of access privilege.

The Requirements column of the following table indicates any special requisites an API function might have. The following list explains what the letters used in that column represent:

W	Requires Requester service
M	Requires Messenger service
S	Requires Server service
R	Can be run remotely
A	Requires Administrative authority (remote only)
P	Requires partial administrative authority
L	Has a local-only library available
c	Requires caller have c authority on each access control list (ACL).

The privileges associated with an API function are listed in the **Operator Rights** column of the table. Operator rights are part of the network accounts file, NET.ACC, and are replicated within the domain. An operator is someone with User authority who also has certain administrative capabilities but does not a full Administrator authority.

An operator can have one or more of the following privileges:

Accounts	Administers user accounts subsystem
Print	Maintains printer queues and devices
Comm	Maintains communication queues and devices
Server	Maintains normal server operations, shared resources, and so on
-	No special privilege required

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
DosBufReset or 32-bit DosResetBuffer	W L	-
DosCallNmPipe or 32-bit DosCallNPIPE	W L	-
DosClose	W L	-
DosConnectNmPipe or 32-bit DosConnectNPIPE	W L	-
DosDeleteMailslot or Dos32DeleteMailslot	W L	-
DosDisconnectNmPipe or 32-bit DosDisconnectNPIPE	W L	-
DosDupHandle	W L	-
DosMailslotInfo or Dos32MailslotInfo	W L	-
DosMakeMailslot or Dos32MakeMailslot	W L	-
DosMakeNmPipe or 32-bit DosCreateNPIPE	W L	-
DosOpen	W L	-
DosPeekMailslot or Dos32PeekMailslot	W L	-
DosPeekNmPipe or 32-bit DosPeekNPIPE	W L	-
DosPrintDestAdd or 32-bit SplCreateDevice	W A	Print
DosPrintDestControl or 32-bit SplControlDevice	W A	Print
DosPrintDestDel or 32-bit SplDeleteDevice	W A	Print
DosPrintDestEnum or 32-bit SplEnumDevice	W	-
DosPrintDestGetInfo or 32-bit SplQueryDevice	W	-
DosPrintDestSetInfo or 32-bit SplSetDevice	W A	Print
DosPrintJobContinue or 32-bit SplReleaseJob	W P	Print
DosPrintJobDel or 32-bit SplDeleteJob	W P	Print
DosPrintJobEnum or 32-bit SplEnumJob	W	-
DosPrintJobGetId	W	-
DosPrintJobGetInfo or 32-bit SplQueryJob	W	-
DosPrintJobPause or 32-bit SplHoldJob	W P	Print
DosPrintJobSetInfo or 32-bit SplSetJob	W P	Print
DosPrintQAdd or 32-bit SplCreateQueue	W A	Print

DosPrintQContinue or 32-bit SplReleaseQueue	W A	Print
DosPrintQDel or 32-bit SplDeleteQueue	W A	Print
DosPrintQEnum or 32-bit SplEnumQueue	W	-
DosPrintQGetInfo or 32-bit SplQueryQueue	W	-
DosPrintQPause or 32-bit SplHoldQueue	W A	Print
DosPrintQPurge or 32-bit SplPurgeQueue	W A	Print
DosPrintQSetInfo or 32-bit SplSetQueue	W A	Print
DosQFHandState or 32-bit DosQueryFHState	W L	-
DosQHandType or 32-bit DosQueryHType	W L	-
DosQNmPHandState or 32-bit DosQueryNPHState	W L	-
DosQNmPipeInfo or 32-bit DosQueryNPipeInfo	W L	-

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
DosQNmPipeSemState or 32-bit DosQueryNPipeSemState	W L	-
DosRead	W L	-
DosReadAsync	W L	-
DosReadMailslot or Dos32ReadMailslot	W L	-
DosSetFHandState or 32-bit DosSetFHState	W L	-
DosSetNmPHandState or 32-bit DosSetNPHState	W L	-
DosSetNmPipeSem or 32-bit DosSetNPipeSem	W L	-
DosTransactNmPipe or 32-bit DosTransactNPipe	W L	-
DosWaitNmPipe or 32-bit DosWaitNPipe	W L	-
DosWrite	W L	-
DosWriteAsync	W L	-
DosWriteMailslot or Dos32WriteMailslot	W L	-
ent_acl_manipulate_apply	C	-
ent_acl_replace_apply	C	-
HPFS386GetInfo16 or HPFS386GetInfo	R S	-
NetAccessAdd or Net32AccessAdd	W R A	-
NetAccessApply or Net32AccessApply	W R A	-

NetAccessCheck or Net32AccessCheck	W	-
NetAccessDel or Net32AccessDel	W R A	-
NetAccessEnum or Net32AccessEnum	W R A	-
NetAccessGetInfo or Net32AccessGetInfo	W R A	-
NetAccessGetUserPerms or Net32AccessGetUserPerms	W R A	-
NetAccessSetInfo or Net32AccessSetInfo	W S R A	-
NetAlertRaise or Net32AlertRaise	W	-
NetAlertStart or Net32AlertStart	W	-
NetAlertStop or Net32AlertStop	W	-
NetAliasAdd or Net32AliasAdd	W R A	Server (*3)
NetAliasDel or Net32AliasDel	W R A	Server (*3)
NetAliasEnum or Net32AliasEnum	W R P	Server (*3)
NetAliasGetInfo or Net32AliasGetInfo	W R P	Server (*3)
NetAliasSetInfo or Net32AliasSetInfo	W R P	Server (*3)
NetAppAdd or Net32AppAdd	W R P	Server
NetAppDel or Net32AppDel	W R P	Server
NetAppEnum or Net32AppEnum	W R P	Server
NetAppGetInfo or Net32AppGetInfo	W R P	Server
NetAppSetInfo or Net32AppSetInfo	W R P	Server
NetAuditClear or Net32AuditClear	W(*5) R A	-
NetAuditRead or Net32AuditRead	W(*5) R A	Accounts, Server, Print, Comm
NetAuditWrite or Net32AuditWrite	W S	-

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
NetBiosClose or NetBios32Close	W	-
NetBiosEnum or NetBios32Enum	W R	-
NetBiosGetInfo or NetBios32GetInfo	W R	-
NetBiosOpen or NetBios32Open	W	-
NetBiosSubmit or NetBios32Submit	W	-

NetCharDevControl or Net32CharDevControl	W S R A	Comm
NetCharDevEnum or Net32CharDevEnum	W S R	-
NetCharDevGetInfo or Net32CharDevGetInfo	W S R	-
NetCharDevQEnum Net32CharDevQEnum	W S R	-
NetCharDevQGetInfo or Net32CharDevQGetInfo	W S R	-
NetCharDevQPurge or Net32CharDevQPurge	W S R A	Comm
NetCharDevQPurgeSelf or Net32CharDevQPurgeSelf	W S R	Comm
NetCharDevQSetInfo or Net32CharDevQSetInfo	W S R A	Comm
NetConfigGet2 or Net32ConfigGet2	W R A	Accounts, Server, Print, Comm
NetConfigGetAll2 or Net32ConfigGetAll2	W R A	Accounts, Server, Print, Comm
NetConnectionEnum or Net32ConnectionEnum	W S R A	Server, Print, Comm
NetCreateRIPLMachine or Net32CreateRIPLMachine	A	Server
NetDASDAdd or Net32DASDAdd	W R A P(*4)	-
NetDASDCheck or Net32DASDCheck	W R A	-
NetDASDCtl or Net32DASDCtl	W R A	-
NetDASDDel or Net32DASDDel	W R A P(*4)	-
NetDASDEnum or Net32DASDEnum	W R A	-
NetDASDGetInfo or Net32DASDGetInfo	W R A	-
NetDASDSetInfo or Net32DASDSetInfo	W R A P(*4)	-
NetDeleteRIPLMachine or Net32DeleteRIPLMachine	A	Server
NetErrorLogClear or Net32ErrorLogClear	R A	-
NetErrorLogRead or Net32ErrorLogRead	R A	-
NetErrorLogWrite or Net32ErrorLogWrite	W	-
NetEnumRIPLMachine or Net32EnumRIPLMachine	A	Server
NetFileClose2 or Net32FileClose2	W S R A	Server
NetFileEnum2 or Net32FileEnum2	W S R A	Server
NetFileGetInfo2 or Net32FileGetInfo2	W S R A	Server
NetGetDCName or Net32GetDCName	W R	-

NetGetRIPLMachine or Net32GetRIPLMachine	A	Server
NetGroupAdd or Net32GroupAdd	W R A	Accounts (*6)
NetGroupAddUser or Net32GroupAddUser	W R A	Accounts (*6)

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
NetGroupDel or Net32GroupDel	W R A	Accounts (*6)
NetGroupDelUser or Net32GroupDelUser	W R A	Accounts (*6)
NetGroupEnum or Net32GroupEnum	W R A	Accounts (*6)
NetGroupGetInfo or Net32GroupGetInfo	W R A	Accounts (*6)
NetGroupGetUsers or Net32GroupGetUsers	W R A	Accounts (*6)
NetGroupSetInfo or Net32GroupSetInfo	W R A	Accounts (*6)
NetGroupSetUsers or Net32GroupSetUsers	W R A	Accounts (*6)
NetHandleGetInfo	W R S	-
NetHandleSetInfo	W R S	-
NetLogonEnum or Net32LogonEnum	W R	-
NetMessageBufferSend or Net32NetMessageBufferSend	W R A	Accounts, Server, Print, Comm
NetMessageFileSend or Net32NetMessageFileSend	W R A	Accounts, Server, Print, Comm
NetMessageLogFileGet or Net32NetMessageLogFileGet	W M R A	-
NetMessageLogFileSet or Net32NetMessageLogFileSet	W M R A	-
NetMessageNameAdd or Net32NetMessageNameAdd	W M R A	-
NetMessageNameDel or Net32NetMessageNameDel	W M R A	-
NetMessageNameEnum or Net32NetMessageNameEnum	W M R A	-
NetMessageNameFwd or Net32NetMessageNameFwd	W M R A	-

NetMessageNameGetInfo or Net32NetMessageNameGetInfo	W M R A	-
NetMessageNameUnFwd or Net32NetMessageNameUnFwd	W M R A	-
NetRemoteCopy or Net32RemoteCopy	W R	-
NetRemoteExec or Net32RemoteExec	W R	-
NetRemoteMove or Net32RemoteMove	W R	-
NetRemoteTOD or Net32RemoteTOD	W R	-
NetServerAdminCommand or Net32ServerAdminCommand	W S R A	-
NetServerDiskEnum or Net32ServerDiskEnum	W R A	Server
NetServerEnum2 or Net32ServerEnum2	W R	-
NetServerGetInfo or Net32ServerGetInfo	W S R A	Accounts, Server, Print, Comm
NetServerSetInfo or Net32ServerSetInfo	W S R A	Server
NetServiceControl or Net32ServiceControl	W R A	Server
NetServiceEnum or Net32ServiceEnum	W R	-
NetServiceGetInfo or Net32ServiceGetInfo	W R	-

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
NetServiceInstall or Net32ServiceInstall	W R A	Server
NetServiceStatus or Net32ServiceStatus	W	-
NetSessionDel or Net32SessionDel	W S R A	Server
NetSessionEnum or Net32SessionEnum	W S R A	Server
NetSessionGetInfo or Net32SessionGetInfo	W S R A	Server
NetSetRIPLMachine or Net32SetRIPLMachine	A	Server
NetShareAdd or Net32ShareAdd	W S R A	Server (*1)
NetShareCheck or Net32ShareCheck	W S R	-
NetShareDel or Net32ShareDel	W S R A	Server (*1)
NetShareEnum or Net32ShareEnum	W S R	Server, Print, Comm
NetShareGetInfo or Net32ShareGetInfo	W S R	Server, Print, Comm

NetShareSetInfo or Net32ShareSetInfo	W S R A	Server (*1)
NetStatisticsGet2 or Net32StatisticsGet2	W R A	Server
NetUseAdd or Net32UseAdd	W R A	-
NetUseDel or Net32UseDel	W R A	-
NetUseEnum or Net32UseEnum	W R A	-
NetUseGetInfo or Net32UseGetInfo	W R A	-
NetUserAdd or Net32UserAdd	W R A	Accounts (*2) (*6)
NetUserDCDBInit or Net32UserDCDBInit	W R P	Accounts
NetUserDel or Net32UserDel	W R A	Accounts (*2) (*6)
NetUserEnum or Net32UserEnum	W R A	Accounts (*6)
NetUserGetAppSel or Net32UserGetAppSel	W R P	Accounts (*6)
NetUserGetGroups or Net32UserGetGroups	W R A	Accounts (*6)
NetUserGetInfo or Net32UserGetInfo	W R A	Accounts (*2) (*6)
NetUserGetLogonAsn or Net32UserGetLogonAsn	W R P	Accounts (*6)
NetUserModalsGet or Net32UserModalsGet	W R A	Accounts (*6)
NetUserModalsSet or Net32UserModalsSet	W R A	Accounts (*6)
NetUserPasswordSet or Net32UserPasswordSet	W S R A	- (*6)
NetUserSetAppSel or Net32UserSetAppSel	W R P	Accounts (*6)
NetUserSetGroups or Net32UserSetGroups	W R A	Accounts (*6)
NetUserSetInfo or Net32UserSetInfo	W R A	Accounts (*2) (*6)
NetUserSetLogonAsn or Net32UserSetLogonAsn	W R P	Accounts (*6)
NetUserValidate2 or Net32UserValidate2	W	-

Table 3. API Requirements and Operator Rights

API NAME	REQUIREMENTS	OPERATOR PRIVILEGES
----------	--------------	------------------------

NetWkstaGetInfo or Net32WkstaGetInfo	W R A	Accounts, Server, Print, Comm
NetWkstaSetInfo or Net32WkstaSetInfo	W R A	-
NetWkstaSetUID2	W	-
SplQmAbort	R	-
SplQmClose	R	-
SplQmEndDoc	R	-
SplQmOpen	R	-
SplQmStartDoc	R	-
SplQmWrite	R	-
UPMELOCL or U32ELOCL	L	-
UPMELOCU or U32ELOCU	L	-
UPMEULGF or U32EULGF	L	-
UPMEULGN or U32EULGN	L	-
UPMEUSRL or U32EUSRL	L	-

FOOTNOTES:

(*1) The print operator can invoke the API on print queues. The comm operator can invoke the API on comm queues.

(*2) The accounts operator cannot invoke this API on an account with administrator privilege, nor can an accounts operator give an existing account administrator privilege. The accounts operator cannot change the operator rights of any account, including creating an account with operator rights.

(*3) The print operator can invoke the API if managing a print alias. A comm operator can invoke the API if managing a serial device alias.

(*4) Nonadministrators can use this API if they have P access privilege to the parent directory.

(*5) The requester must be running, but no one has to be logged on for this function.

(*6) There are no accounts operators in DSS. Authority is determined by ACLs on the object being acted on.

Migration Compiling

DosAllocMem allocates nonshared memory, so you do not need to define SEG_NONSHARED in 32-bit code. Make sure you compile using the /Sm option if your C file uses any migration functions.

Install the migration library and header files when you install the IBM C Set/2 compiler or equivalent compiler. (The FCNTL.H header file is installed when you install the migration support with C Set/2.)

Protection Violations and Faults in Dynamic Link Libraries

The API functions probe the buffers passed to them and scan string parameters in an attempt to ensure that the data is accessible. These probes can cause faults if the pointers are incorrect (for example, if they are pointing beyond the end of a segment or outside a permitted memory region).

If you get a fault within a LAN Server dynamic link library, attempt to trace the code through the call. By noting the values that are being tested, you usually can recognize the parameter that is causing the problem. Also, check buffer sizes carefully, because the API functions probe the first and last byte of a buffer even if the data returned or received does not fill the buffer.

If you get a stack overflow, extend the stack size. There is no absolute rule for determining the depth of a stack that a LAN Server API function requires. Generally, allow 4KB of free stack space for each function call.

DOS LAN Services

This section presents information programmers should be aware of when using LAN Server APIs with DOS LAN Services (DLS).

In this section, you will find information about:

- IBM DLS API services
- IBM DLS libraries, where they are stored, and which ones are needed for particular APIs
- API functions supported under IBM DLS and any differences in their use

Note: A DLS peer workstation does not participate in any OS/2 LAN Server domain. It works like an OS/2 server in a role of standalone or an OS/2 Peer workstation. Also, it cannot be administered by using the NET ADMIN command from a remote node. However, a user can administer a DLS peer workstation remotely from another DLS workstation or an OS/2 requester or server. When porting LAN Server applications to run under DLS, be aware that DOS, unlike the OS/2 operating system, does not support pointer checking, semaphores, or shared memory segments. All file names, directory names, or parts of a path name, including UNC server and share names, must follow DOS naming conventions.

Most of the LAN Server APIs are accessible from a DLS workstation. Except in several cases, such as the Handle, Mailslot, Named Pipe, and other DOS-supported APIs, calls to the APIs described in this book can be issued from a DLS workstation only to remote OS/2 LAN servers.

Services Supported under DLS

DOS LAN Services supports the following services, which are installed during installation of the redirector program.

SERVICE	PURPOSE
Messenger	Enables DLS workstations to send and receive messages across the LAN
Netpopup	Enables messages to be displayed as pop-up messages on the screen
Requester	Enables DOS computers (attached to the LAN) to be configured as requesters, thus enabling them to access resources on remote OS/2 LAN Servers

Peer

Enables DLS workstations to share printers and drives

Linking to API Libraries for DLS

An OS/2 program uses dynamic link libraries during run time, where some of the functions are called like subroutines, allowing more function with fewer memory requirements. Although DOS does not use dynamic link libraries, Microsoft Windows 3.1 does.

LAN Server provides dynamic link libraries, NETAPI.DLL and PMSPL.DLL, for use with Windows 3.1. In order to call LAN Server APIs, a Windows application must link to the NETAPI.LIB and PMSPL.LIB (for DosPrint API support) libraries.

Link a DOS application to the DOSNET.LIB library. This library file provides the network functions for applications on DLS workstations to access LAN Server.

Link your application to these .LIB files during compile with a compiler such as Microsoft C Version 6.0 or later, Microsoft C/C++ Version 7.0 or later, or Microsoft Visual C++ Professional Edition Version 1.5 or later (which supports the LAN Server DOSNET.LIB library for DLS applications), Borland C/C++ Version 4.0 or later.

Depending on the compiler and your options, the command to link a DOS application to the API libraries might resemble this:

- For non-Windows, DLS (DOS) applications:

```
LINK MYDOSAPP.OBJ, MYDOSAPP.EXE, MYDOSAPP.MAP,  
d:DOSNET.LIB
```

- For Windows applications:

```
LINK MYWINAPP.OBJ, MYWINAPP.EXE, MYWINAPP.MAP,  
d:NETAPI.LIB  
d:PMSPL.LIB
```

where *d:* includes the drive and directory path for the .LIB file. The DOSNET.LIB, NETAPI.LIB, and PMSPL.LIB libraries are located by default in the \NET directory on the workstation. If you have remote IPL support on a server, the library files are stored by default in the \BMLAN\DOSLAN\NET subdirectory of that server. (Refer to the documentation accompanying your compiler for instructions and options.)

Header Files

DLS applications use most of the same header files as an OS/2 application. LAN Server also provides DLS-specific header files, DOSPRINT.H and NMPIPE.H, to support DLS application calls to remote print and named pipe functions. These files are stored by default in the \BMLAN\NETSRCH subdirectory.

For all the Print APIs, DLS applications should use the DOSPRINT.H header file instead of PMSPL.H. Also, you do not need to include the OS2.H header file or #define INCL_BASE. For example, use:

```
#include <dosprint.h>
```

for DLS applications instead of:

```
#define INCL_BASE  
#include <os2.h>  
#include <pmspl.h>
```

which is used only for 16-bit OS/2 applications.

Note: Simultaneous use of OS2DEF.H and WINDOWS.H header files produces conflicts resolved by constant and type redefinitions.

Using Remote Procedure Calls

The Remote Procedure Call (RPC) component gives the administrator and application developer very precise control of the level of security used for making RPCs. However, security has its cost, so it is recommended that the application use the minimum level of security that meets requirements. Integrity and privacy levels can be very costly for large data.

If you are handling a series of RPC calls over time, you can increase speed by using a context handle, which causes the RPC runtime to keep an active connection between client and server. This avoids the cost of connection setup and teardown, typically done after about 10 seconds of inactivity.

There are some differences in performance characteristics of the different protocol sequences and their associated transports. Connection-oriented (CN) is quicker to detect a down server, but does not scale as well as datagram (DG), because the server needs a separate connection and connection receiver thread for each concurrently active client. At small data sizes CN and DG have similar performance, but CN is slightly faster at large data sizes.

Using Interface Definition Language

When you define the interfaces to your application using the Interface Definition Language (IDL), you can minimize the time spent marshalling and unmarshalling passed parameters. Use simple structures and avoid pointers. If an RPC returns a pointer as an out parameter, the client-side unmarshalling routines needs to allocate memory for that parameter. Use reference pointers rather than full pointers when possible. In many cases reference pointers do not require memory allocation. You can reduce the number of pointers by flattening your data structures, replacing a pointer to an object with an instance of the object instead.

Do not transfer anything you do not really need. Use different data structures for different calls rather than complex one-size-fits-all structures with fields that many calls do not use. Instead of returning data back from one call to be passed back in subsequent calls, think about caching that data in a context handle. Use variable length conformant arrays instead of maximum-sized fixed length arrays (for example, 1024-byte name fields sized to handle the maximum possible name). In addition, if the client connects to the server through a slow link, for example a dial-up link, it takes a lot longer to send those extra bytes across the wire.

Investigate using *transmit_as* to turn large complex structures into small opaque ones for faster marshalling.

Using Naming Services

Binding to a server is an investment of time that you want to use carefully. Bind once; call often. Cache server bindings and reuse them as much as possible. Do not get a new binding each time for an RPC, to then throw away when the call is done. While it may be easier to code library routines and object methods that way, it can make a difference in your application's performance.

Structuring a name space can also impact binding performance. Profiles and groups add a lot of flexibility, but it takes longer to fetch the members of a group or profile than it does to read the contents of a server entry. For each member of the group or profile, the contents of that member have to be processed. If you have a deep hierarchy of profiles and groups, the binding code could check lot of entries before it finds one that matches the binding request. A strategy to consider is to start with a single well-known server name, and then try a group or profile if that fails.

CDS provides for client-side caching of directory information, and that caching can dramatically improve binding performance. Do not disable caching unless it is absolutely necessary. The best strategy is to optimize for the successful case. Do initial lookup with caching enabled. If the call fails using the returned binding, go back and do the lookup with caching disabled to get the latest up-to-date server bindings and repeat the call.

Using Security

The RPC runtime caches call-related information so subsequent calls are be faster than the first. This cached call information is keyed by the identity associated with the binding, so if you change the authority information on the binding, the cached call is not reused for the next call. If your application switches back and forth between a few identities, you can improve performance by keeping a separate binding for each identity, instead of resetting the same binding.

For the same reason you want to conserve the identities that you use. Delegation creates new identities. These identities are several KBs in size and are cached for about a minute. The rate at which your application creates identities determines how many, on average, are in the cache.

DSS Issues

DSS stores aliases in the cell directory, and loads them at logon time. Reducing the aliases associated with a user can reduce the time it takes for the user to log on. The main use that DSS makes of the security and directory services is at logon and net use time. **

General DCE Performance Suggestions

If your server does a lot of lookups from the backing store routines, you might speed it up by loading the whole database into memory at startup time, trading off space for time. This of course assumes that the database fits into memory and you have memory to spare.

A functional problem can sometimes appear to be a performance problem. If you have replicated servers and one of the servers goes down, clients may timeout trying to talk to the dead server before trying the functioning server. This gives the appearance that the system is working slowly.

While RPCs continue to work at low speeds, large amounts of data still take a while to get across the link. If you expect your application to be operating in a low-speed environment, it is very important to follow the suggestions given above for reducing marshalling time. Only exchange the minimum amount of data necessary to get the job done.

Remember to free memory returned by DCE APIs. Over time, a small memory leak in a server can build up to a large amount of wasted swapper space, and cause performance degradation of the system as a whole, as more time is spent swapping memory in and out from disk, and less time spent doing useful work.

Many of the data structures that APIs return have their own free routines (for example, **rpc_string_free**) while others are just freed by the generic **dce_free**. Also be aware of when you should be using **rpc_ss_mem_free**.

Also, the sizing of the application servers depends on the types of transactions. For example, compare the TPC-A benchmark, where speeds are expressed in transactions per second, to the TPC-C benchmark, where speeds are expressed in transactions per minute. A server can support several more TPC-A clients than TPC-C clients.

Dynamic sizing takes several steps. First you need to determine your application's load on the security and CDS servers, then decide the server resources required. Because CDS keeps its directory in memory, it should be CPU-bound and scale about the same as standard benchmarks. The security server tends to have a lot less interaction with applications.

Second, you need to measure your application server, to find out what throughput it can sustain. This depends on the application itself. If you can do a comparison of its requirements to the TPC benchmarks, you might be able to come up with a scaling factor to use, for example, for TPC-A numbers that you could use to estimate the capacity of different machines based on their published TPC-A numbers. Keep in mind that the TPC benchmark numbers are probably much higher than the typical application, because vendors put much effort into ensuring that their systems can get high TPC numbers.

Using Threads

Threads are an important DCE feature that can affect DSS performance. It is recommended to conserve threads whenever possible. Although threads are convenient for structuring an application, they have a cost. Each thread has its own stack and associated data structures that add to the memory requirements of the application. In addition, threads are usually scheduled in round-robin fashion, with each thread waiting its turn to run. Thus, the more active threads, the longer a given thread may have to wait before it runs.

Also, creating and destroying threads takes time. Instead of creating a thread whenever needed and destroying it when the job is done,

consider using a thread pooling strategy. This way, threads that have finished their assignment check a queue of jobs for more work, and wait if no work is available.

Because a thread can be interrupted at any moment and another one scheduled, try to avoid holding locks unnecessarily. If a thread is interrupted while it holds an important lock and another thread is dispatched, only to try to grab the lock and get blocked, time has been wasted on context switching back and forth. The second thread is not going to make any progress until the first one releases the lock. You can reduce this occurrence by holding locks for shorter times, and by using finer grained locks. Use different locks for logically separate structures rather than one global lock. However, you have to trade this against the cost of locking and unlocking.

Sample Programs

This chapter presents sample programs for LAN Server. These programs are applicable to Warp Server and DSS also. Included in this chapter is a section on Password Strength Server coding.

Sample Programs for LAN Server

OS/2 LAN Server provides sample application resource files that you can open and view, print, or possibly use as segments in your own program. When you install the IBM Developer's Toolkit for OS/2 Warp Version 4 installation program, these sample files are copied by default into the \TOOLKIT\SAMPLES\LANSERV subdirectory.

These sample programs are written in 32-bit code, except for MLOGON32.C, which is a mixed-modal application that includes both 16-bit and 32-bit API calls. (MLOGON32.C logs a user on and off a domain.)

Note: For MOCL sample programs, see *IBM DCE for OS/2: Managed Object Class Library Programmer's Guide and Reference*. For DCE sample programs, see *IBM DCE for OS/2 Warp: Application Development Reference*.

ACCESS32.C

This program displays or sets access control information about a resource specified on the command line.

In this application, the following APIs are called:

```
Net32AccessAdd
Net32AccessGetInfo
Net32AccessSetInfo
```

ALIAS32.C

This program takes an alias name and a resource name from the command line and performs the following actions:

- Adds the alias definition
- Shares the resource
- Adds access control to the resource
- Attempts to add a directory limit to the resource

In this application, the following APIs are called:

```
Net32AccessAdd
Net32AccessGetInfo
```


Net32AccessSetInfo
Net32AliasAdd
Net32DASDAdd
Net32GetDCName
Net32ServerEnum2
Net32ShareAdd
Net32WkstaGetInfo

FILEEN32.C

This program displays opened files on a particular server. The *pszServername*, *pszUserID*, and *pszPath* can be specified optionally at the command line as qualifiers for the set of files to be enumerated. If none of these parameters is specified, this program displays all of the open files on the local server.

This program also demonstrates the *pResumeKey*, which is used to retrieve more than 64KB (the limit to a single call) of data from an API. The *pResumeKey* allows multiple calls in succession to the API, where the API returns up to 64KB with each call, beginning where the previous call ended. (Enum APIs often allow successive calls.)

In this application, the following API is called:

Net32FileEnum2

MLOGON32.C Mixed-Modal: 16-Bit and 32-Bit

Using a mixed-modal format, which includes both 16-bit and 32-bit API calls, this program logs a user on and off a domain.

In this application, the following APIs are called:

UPMEULGF
UPMEULGN

MESSAG32.C

This program prompts the user for a message to send and the user ID to send it to (including the name of the server to which the recipient is logged on). Then it sends the message.

In this application, the following API is called:

Net32MessageBufferSend

SADMIN32.C

This program runs a command on a remote server and displays the command's output. SADMIN32.C is similar in function to the NET ADMIN command.

In this application, the following API is called:

Net32ServerAdminCommand

USER32.C

This program performs the following actions:

- Displays details of a user
- Deletes a user definition
- Adds a user definition
- Creates a new group
- Adds a user to a group
- Adds an alias
- Gives a user access to an alias
- Assigns an alias to a user as a logon assignment
- Creates a private application
- Assigns a private application to a user

In this application, the following APIs are called:

```
Net32AccessAdd
Net32AccessGetInfo
Net32AccessSetInfo
Net32AliasAdd
Net32AliasDel
Net32AliasGetInfo
Net32AppAdd
Net32GetDCName
Net32GroupAdd
Net32GroupAddUser
Net32GroupDel
Net32ServerEnum2
Net32UserAdd
Net32UserDCDBInit
Net32UserDel
Net32UserGetGroups
Net32UserGetInfo
Net32UserGetLogonAsn
Net32UserSetAppSel
Net32UserSetLogonAsn
Net32WkstaGetInfo
```

Password Strength Server

A password strength server (PSS) ensures that newly created or modified passwords follow specified rules of password composition. The password strength server (if active) installed with DSS tests new passwords (created or modified) and forces them to conform to the password composition rules.

If your DSS installation requires password composition rules other than those provided you can modify and recompile the PSS daemon to customize the rules. The password strength server that ships with DSS allows changes to the customer composition and history.

Customers should tailor the following parameters when creating their own password strength server:

```
SERVER_PNAME
OBJUUID
```

SERVER_PNAME will be the unique identity of each potentially available strength server running within the cell. There can be multiple strength servers running in a cell at the same time, however, only ONE of these strength servers can be assigned to each user account at a time. For DSS,

lspwds

is the default assignment. When creating a new strength server from this code, you must compile under this name. This is the name of the error log for this server.

OBUUID is a string that is the unique identifier for this server. The OBUUID allows the server to communicate with multiple parts of DSS. The string **MUST** be changed for each new strength server to run in any cell. The attain a new uuid for this parameter:

Go to an active DCE session window.

Type: uuidgen

Press enter.

A new and unique uuid will be returned. Replace the current OBJUUID with the new uuid.

The only other location in the code that a customer needs to alter is the actual password composition check. To find this location in the code search for the string

CUSTOMER COMPOSITION CHECKING

The following code is contained in the

lspwsc.c

header information:

```
#define SERVER_PNAME "lspwspd"      /* server principal name, program name
                                     (.exe name), and error log name */
#define OBJUUID "9bdda41-1940-11cf-8f92-000233002ab7" /* object uuid */
#define CDSPATH "/./:/subsys/dce/pwd mgmt/" SERVER_PNAME /* export bindings */
```

All strength servers, whether provided by IBM or built by customers, should use the principal name as a "simple" suffix to the CDS namespace path. This convention must be held to if OS/2 DCE configuration procedures in support of strength servers are to function properly and conveniently for you.

The password strength server has been modified to:

- Call routine `strength_unregister()` to do most exit cleanup. One reason for isolating this into a separate routine is to allow OS/2 DCE an option on where and how deregistration occurs.
- Unexport information in the CDS namespace during initialization. This eases transition in the event that a strength server is to be located to a different machine. Server keytable must be moved manually, but CDS information can be handled by code.
- Unexport information in the CDS namespace during initialization in case the strength server shutdown from the previous run did not go cleanly.

Use the "cds style" way of catching signals and terminating. The essential features of this style are:

- Separate thread to do `rpc_server_listen`
- Use of mutex and condition variable between signal catcher and main thread.
- Elimination of `TRY/ENDTRY` macros, etc.

The following code is contained in the

rser_pwd_mgmt_str_chk

•

[illegible]

```

}
if(!(password_validity))
{
    if((*stp == NERR_PasswordTooRecent)
        ||
        (*stp == NERR_PasswordHistConflict)
        ||
        (*stp == NERR_PasswordTooShort)
        ||
        (*stp == NERR_AllAlphaNum)
        ||
        (*stp == NERR_PasswordAllSpaces))
    {

/* Call DosGetMessage to get the message from the message file. Ignore */
/* the return code because if there is an error, the text for that error */
/* is returned in msg_buf. */

        DosGetMessage(NULL,          /* pointer to substitution strings */
            0,                      /* number of substitution strings */
            msgbuf,                  /* return buffer */
            MSGBUF_LEN,
            *stp,                    /* message number */
            "net.msg"                /* message file name */
            &msglen);                /* length of returned message */
        strcpyf(net_msgbuf,MSG_HEADER);
        strcatf(net_msgbuf,prin);
        strcatf(net_msgbuf,' ' );
        switch(* stp)
        {
            case NERR_PasswordTooRecent:
                strcatf(net_msgbuf,"NERR_PasswordTooRecent ");
                break;
            case NERR_PasswordHistConflict:
                strcatf(net_msgbuf,"NERR_PasswordHistConflict");
                break;
            case NERR_PasswordTooShort:
                strcatf(net_msgbuf,"NERR_PasswordTooShort ");
                break;
            case NERR_PasswordAllAlphaNum:
                strcatf(net_msgbuf,"NERR_PasswordAllAlphaNum");
                break;
            case NERR_PasswordAllSpaces:
                strcatf(net_msgbuf,"NERR_PasswordAllSpaces");
                break;
        }
        strcatf(net_msgbuf,msgbuf);

        Net32ErrorLogWrite (NULL,
            stp,
            (const unsigned char *)"LSPWSD"
            (const unsigned char *)net_msgbuf,
            sizeof(net_msgbuf),
            NULL,
            0,
            NULL);

        DSSgetmessage;
    }
}

/* THIS PASSWORD COMPOSITION CHECK **MUST** RETURN **ONLY** A TRUE OR */
/* FALSE AS TO PASSWORD VALIDITY */

return (bollean32)password_validity;
}

```

Note: Replace the OBJUUID with UUID Gen on command line. This will generate a unique object id.

For more information concerning Password Strength Server, refer to *IBM Directory and Security Server Administrator's Reference*.

Access Permission Category

This category includes the following APIs:

- Access Permission - NetAccessAdd
- Access Permission - NetAccessApply
- Access Permission - NetAccessCheck
- Access Permission - NetAccessDel
- Access Permission - NetAccessEnum
- Access Permission - NetAccessGetInfo
- Access Permission - NetAccessGetUserPerms
- Access Permission - NetAccessSetInfo
- Access Permission - ent_acl_replace_apply
- Access Permission - ent_acl_manipulate_apply

Access permission APIs examine or change user or group access permission records for server resources. They are used with the ACCESS.H and NETCONS.H header files.

Note: The following five APIs are the only ones supported by DSS servers. DSS servers use the DCE access control model. There is only limited support for the NetAccess APIs. DSS clients can remote all APIs to LAN Server or Warp Server servers.

- NetAccessApply or Net32AccessApply
- NetAccessCheck or Net32AccessCheck
- NetAccessGetUserPerms or Net32AccessGetUserPerms
- ent_acl_replace_apply
- ent_acl_manipulate_apply

To set access permission on DSS servers, you must use either the DCE sec_acl APIs or MOCL. See:

- *IBM DCE for OS/2 Warp: Application Development Guide - Core Components*
- *IBM DCE for OS/2: MOCL Programmer's Guide and Reference*

For a user to access a shared resource, an *access permission record* must be defined for that resource. An access permission record defines how a user or group can access a shared resource. It contains a set of permissions for each user or group.

Access permission records are created with the NetAccessAdd API. To delete all access permission records associated with a particular shared resource, call the NetAccessDel API.

An access permission record contains:

- The name of the resource
- A list of users or groups and their access permissions
- The audit trail flag, which controls whether audit trail records are written for access to the resource

The NetAccessGetInfo API can be called to return information about a particular access permission record. To obtain information about all access permission records for which the calling process has special permissions (ACCESS_PERM), call the NetAccessEnum API.

The access permission record must be defined by a user or an application that already has either administrative permissions or special permission (ACCESS_PERM) for the resource being shared. User permissions have precedence over group permissions. If a user is not defined in the access list for the shared resource, the user's access permissions are the union of all groups to which the user belongs. For more information about access control checking, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.

Note: For MOCL programs see *DCE for OS/2: Managed Object Class Library Programmer's Guide and Reference*. For DCE programs, see *DCE for OS/2 Warp: Application Development Reference*.

DSS Remote Procedure Calls

Remote procedure calls (RPC) allow an access control list (ACL) to be propagated within a directory or down an entire subtree. Two RPCs are added to the Access Permission Category: *ent_acl_replace_apply* and *ent_acl_manipulate_apply*.

DOS Considerations

Under DOS, the access permission APIs can be run only on a remote server. In using NetAccessEnum when level 1 is selected, the Auditing bit (*USHORT acc_attr*) does not have any meaning as it is not supported by any DLS workstation.

Resource Access Data Structures

The *level* parameter controls the level of information provided to or returned from the NetAccessAdd, NetAccessEnum, NetAccessGetInfo, and NetAccessSetInfo APIs. These APIs use either a level 0 or a level 1 data structure.

Resource Access Level 0

```
struct access_info_0 {
    unsigned char LSFAR * LSPTR    acc0_resource_name;
};
```

where:

- acc0_resource_name* points to an ASCIIZ string containing the name of a resource type. *acc0_resource_name* uses the following formats:

RESOURCE TYPE	NAME FORMAT
Drive	drive:
Directory	drive:pathname
File	drive:pathname\filename
Pipe	\PIPE\pipename
Spooler queue	\PRINT \queuename
Serial device queue	\COMM\chardevqueue

Resource Access Level 1

```
struct access_info_1 {
    unsigned char LSFAR * LSPTR    accl_resource_name;
    short            accl_attr;
    short            accl_count;
```

```
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *acc1_attr* specifies the attributes of *acc1_resource_name*. The bits of *acc1_attr* are defined as follows:

BIT	MEANING
0	Audit all. When this bit is set, all access attempts are audited. No other bits in the field can be set. It is an error to set any other bits when bit 0 is set. When bit 0 is cleared, the remaining bits are defined as described in this table.
1-3	Reserved with a value of 0.
4	If 1, audit successful file opens.
5	If 1, audit successful file writes and successful directory creates.
6	If 1, audit successful file deletes or truncates and successful directory deletes.
7	If 1, audit successful file and directory access control profile changes.
8	If 1, audit failed file opens.
9	If 1, audit failed file writes and failed directory creates.
10	If 1, audit failed file deletes or truncates and failed directory deletes.
11	If 1, audit failed file and directory access control profile changes.
12-15	Reserved with a value of 0.

NOTES:

Other resources that can be accessed across the network, including spooler queues, serial device queues, and pipes, are audited using the FOR FILES bits.

A value of 0 for the *accl_attr* word means there is no auditing of resource accesses. A value of 1 means audit everything. Other values indicate the auditing of specific accesses.

When write auditing is enabled, the write audit record is generated when the file is opened successfully for write. Only one write audit record is produced for each open instance of the file. If both write and open auditing are enabled, two audit records can be produced.

File size changes (including truncation) are audited under the control of auditing bits 5 and 9. Thus, access that is controlled with the ACCESS_WRITE permission bits is audited by way of auditing bits 5 and 9.

Bit 3 is used in conjunction with bit 4 to allow the auditor to determine the duration of access. However, because this information is not required, the generation of the close audit is optional.

- *acc1_count* specifies the number of *access_list* data structures following the *access_info_1* data structure.

In addition, the *access_info_1* data structure can be followed by 0-64 *access_list* data structures. These structures define resource permissions for individual users or groups.

User Access List Data Structure

```
struct access_list {
    unsigned char  acl_ugname[UNLEN+1];
    unsigned char  acl_ugname_pad_1;
    short          acl_access;
};
```

where:

- *acl_ugname* is an ASCIIZ string specifying a particular user name or group name.
- *acl_ugname_pad_1* word-aligns the data structure components.
- *acl_access* specifies permission of a user name or a group name. *acl_access* is defined in the ACCESS.H header file as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
ACCESS_NONE	0	No permission to access the resource.
ACCESS_READ	0x01	Permission to read data from a resource and, by default, run the resource.
ACCESS_WRITE	0x02	Permission to write data to the resource.
ACCESS_CREATE	0x04	Permission to create an instance of the resource (such as a file); data can be written to the resource when the resource is being created.
ACCESS_EXEC	0x08	Permission to run the resource.
ACCESS_DELETE	0x10	Permission to delete the resource.
ACCESS_ATTRIB	0x20	Permission to change the attributes of a resource (such as the date and time a file was last changed).
ACCESS_PERM	0x40	Permission to change the permissions (read, write, create, execute, and delete) assigned to a resource for a user or application.
ACCESS_ALL	0x7F	Permission to read, write, create, execute, or delete a resource, or to change attributes or permissions.
ACCESS_GROUP	0x8000	Permission for a particular group; if returned, indicates that the entry is for a group.

Apply Access Control Data Structure

```
struct applyerr_info_1 {
    unsigned char LSFAR * LSPTR    error_buf;
    unsigned short                                error_code;
};
```

where

- *error_buf* points to an ASCIIZ string specifying the complete path name of the resource that contributed to the failure of the apply function.
- *error_code* specifies the code returned when the ACL for *error_buf* resource is changed.

Apply Remote Procedure Call Data Structure

```
typedef struct {
    boolean fOldIsLocal;
    union {
        sec_id_t          local;          //TRUE
        sec_id_foreign_t  foreign;        //FALSE
    } old_id;

    boolean fNewIsLocal;
    union {
        sec_id_t          local;          //TRUE
        sec_id_foreign_t  foreign;        //FALSE
    } new_id;
} id_to_replace_t;
```

where

- *fOldIsLocal* is the discriminator for union *old_id*.
- *old_id.local* is the id of the local user or local group that will be renamed.
- *old_id.foreign* is the id of the foreign user or foreign group that will be renamed.
- *fNewIsLocal* is the discriminator for union *new_id*.
- *new_id.local* is the id of the new local user or local group.
- *new_id.foreign* is the id of the new foreign user or foreign group.

```
typedef struct {
    unsigned32 action;
    union {
        id_to_replace_t user;          //ENTRY_RENAME_USER
        id_to_replace_t group;         //ENTRY_RENAME_GROUP
        sec_acl_entry_t sec_acl;       //all other actions
    } target;
} ent_entry_to_manipulate_t;
```

where

- *action* specifies what action to take on the *target*. Valid action codes are:

SYMBOLIC CONSTANT	MEANING
ENTRY_RENAME_USER	Changes the name of the specified user to a new name. Only the UUID is required, but if the UUID is the NIL_UUID, the RPC will resolve the name to its UUID.
ENTRY_RENAME_GROUP	Same as ENTRY_RENAME_USER except that it works on group entries.
ENTRY_CREATE	Add entry to ACL if it does not exist. If the entry exists, do nothing (no permission bits will be modified).
ENTRY_DELETE	Delete entry from ACL if it exists. If the entry does not exist, do nothing.
ENTRY_MODIFY_ADD	If the entry exists in the ACL, this set of permission bits are turned on (perm_bits = specified_perm_bits) If the entry does not exist, do nothing.
ENTRY_MODIFY_ADD_OR_CREATE	If the entry exists in the ACL, execute an ENTRY_MODIFY_ADD. If not, perform an ENTRY_CREATE.
ENTRY_MODIFY_DELETE	If the entry exists in the ACL, this set of permission bits are turned off (perm_bits &= ~specified_perm_bits) If the entry does not exist, do nothing.
ENTRY_REPLACE	If the entry exists in the ACL, replace the existing permission bits with the specified set (perm_bits = specified_perm_bits) If the entry does not exist, do nothing.
ENTRY_REPLACE_OR_CREATE	If the entry exists in the ACL, execute an ENTRY_REPLACE. If not, perform an ENTRY_CREATE.

Note: If a foreign user is specified, then both parts of the foreign id (cell and user) must be in the same format. Both parts must be in either UUID or name form.

It is allowed to have the old name and the new name in different formats.

- *action* is also the discriminator for union *target*.
- *target* is a union of *id_to_replace_t* and *sec_acl_entry_t*. All actions use *sec_acl_entry_t* except *ENTRY_RENAME_USER* and *ENTRY_RENAME_GROUP*.

```
typedef ent_entry_to_manipulate *ent_entries_to_manipulate_p_t;
```

```
typedef struct {
    unsigned32          num_entries_to_change;
```

```

        ent_entry_to_manipulate_t entries_to_change_list[1];
    } ent_entries_to_manipulate_t;

```

where

- *num_entries_to_change* specifies how many entries are in the array *entries_to_change_list*.
- *entries_to_change_list* is the first element in an array of *ent_entry_to_manipulate_t* structures.

```

typedef struct {
    char          *resource_name;
    error_status_t error_code;
} error_entry_t;

```

where

- *resource name* points to the path name of the resource being processed when the error occurred during apply.
Note: The client must free the memory used by calling *rpc_sm_free()*.
- *error_code* is the error status code.

NetAccessAdd or Net32AccessAdd

NetAccessAdd or Net32AccessAdd

The NetAccessAdd API is used to add a new access permission record to a particular resource. This record specifies the level of access permission to be granted to a user or group ID for that resource.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

If this API is issued to a DSS server, NERR_ResourceExists will be returned.

Syntax

```

#include <netcons.h>
#include <access.h>

NetAccessAdd(pszServername, sLevel, buf, usBuflen);      /* 16 bit */
Net32AccessAdd(pszServername, ulLevel, buf, ulBuflen);  /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in [Resource Access Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ResourceExists	2225	The resource permission list already exists.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_ACFTooManyLists	2230	Too many lists were specified.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.

NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
------------------------	------	---

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosQFileMode

Remarks

To define the access permissions for a new resource, the contents of *buf* must include an *access_info_1* data structure specifying the name of a resource, its attributes, and the number of *access_list* data structures that are appended. Each *access_list* data structure specifies a user name or group name and associated permissions to be added to the access permission record of a resource.

Related Information

For information about:

- Deleting an access permission record, see [Access Permission - NetAccessDel](#).
- Listing server permissions and resources, see [Access Permission - NetAccessEnum](#).

NetAccessApply or Net32AccessApply

NetAccessApply or Net32AccessApply

After an access permission record is defined for a directory, the NetAccessApply API replicates that record, applying it to all subdirectories under that directory.

This API also updates the access permission for any file within the directory tree that already has an access permission record. It does not create a record for a file that does not already have one. (If it does not have its own access permission record, a file defaults to the record of its parent directory.) This API does not replace existing audit trail settings, but it does apply the audit settings of the specified directory to any of its subdirectories that have no audit settings.

Restrictions

This API does not apply an access permission record to any of the following LAN Server system subdirectories:

- \BMLAN
- \BMLAN\DCDB and its subdirectories
- \BMLAN\NETPROG and its subdirectories
- \BMLAN\DOSLAN and its subdirectories

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Directory and Security Server Only

NetAccessApply must be run from a DSS server or client. If it is run from a LAN Server or OS/2 Warp Server server, this API returns NERR_InvalidAPI.

Syntax

```
#include <netcons.h>
#include <access.h>
```

```
NetAccessApply(pszServername, pszDirPath, buf, usBuflen,
               reserved1, reserved2, options);    /* 16 bit */

Net32AccessApply(pszServername, pszDirPath, buf, ulBuflen,
                 reserved1, reserved2, options);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

buf (unsigned char LSFAR *) points to the data structure described in [Apply Access Control Data Structure](#).

reserved1 (unsigned char LSFAR *) must be NULL.

reserved2 (unsigned long) must be 0.

options (unsigned long) must be APPLY_ABSOLUTE (value of 0x01), which replaces any existing access control profile for all subdirectories under the specified directory.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_BAD_PATHNAME	161	Path name not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_UnknownDevDir	2116	The device or directory does not exist.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_ApplyNotPermitted	2800	The apply operation is not allowed on the specified path.
NERR_IncompleteApply	2801	The apply operation ended prematurely.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ApptyUnexpectedError	7799	The apply server encountered an unexpected

error.

Other codes could be returned from the following functions:

- DosExecPgm
- DosAllocShrSeg
- DosFsCtl

NetAccessCheck or Net32AccessCheck

NetAccessCheck or Net32AccessCheck

The NetAccessCheck API verifies that a user has the proper access permission for a particular resource.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from OS/2 servers. OS/2 workstations can issue this call locally only if the Peer service is started.

Directory and Security Server Only

This API returns an ERROR_ACCESS_DENIED if the user is not logged on.

A redirected drive should not be specified for *pszResource*. If a redirected drive is specified, a NERR_ResourceNotFound error code is returned.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessCheck(reserved, pszUserID, pszResource,
               AccessPermission, pResult);    /* 16 bit */

Net32AccessCheck(reserved, pszUserID, pszResource,
                 AccessPermission, pResult);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

reserved (unsigned char LSFAR *) must be NULL.

AccessPermission (16-bit unsigned short or 32-bit unsigned long) specifies which access permission the API is to verify for this user on this resource. Defined in the ACCESS.H header file, any combination of the following values can be used:

SYMBOLIC CONSTANT	VALUE	MEANING
ACCESS_READ	0x01	Permission to read data from a resource. By default, run the resource.
ACCESS_WRITE	0x02	Permission to write data to the resource.
ACCESS_CREATE	0x04	Permission to create an instance of a resource (such as a file) data to the resource when the resource is not open.

created.

ACCESS_EXEC	0x08	Permission to run the resource.
ACCESS_DELETE	0x10	Permission to delete the resource.
ACCESS_ATTRIB	0x20	Permission to modify the attributes of the resource (such as the date and time the resource was last modified).
ACCESS_PERM	0x40	Permission to modify the permissions of the resource (read, write, create, delete, execute, and delete permissions).
ACCESS_ALL	0x7F	Permission to read, write, create, delete, or delete a resource, or to modify permissions.

pResult (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer specifying whether the requested access is permitted. Access is permitted only if *pResult* points to 0, and the return code from the API is 0 (NERR_Success).

Return Codes

The following table lists the return codes most significant to this API. See [API Return Codes](#) for a complete list of return codes.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.

Other codes could be returned from the following functions:

- DosAllocSeg

- DosChgFilePtr
- DosFsCtl
- DosOpen
- DosRead

Remarks

If an access permission record cannot be found for the specified user name and the specified resource, the NetAccessCheck API tries to find the proper access permission record for the *GUEST account*, a special account set up for temporary users of the resource. GUEST accounts are defined in the IBMLAN.INI file.

Related Information

For information about:

- Defining user or group access permissions, see [Access Permission - NetAccessAdd](#).
- GUEST accounts, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Listing all permissions and resources, see [Access Permission - NetAccessEnum](#).

NetAccessDel or Net32AccessDel

NetAccessDel or Net32AccessDel

The NetAccessDel API deletes all access permission records for a particular shared resource.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started.

Administrator authority is required to call this API.

Directory and Security Server Only

When this API is remoted or issued by a DLS workstation to a DSS server, ERROR_ACCESS_DENIED is returned.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessDel(pszServername, pszResource);          /* 16 bit */
Net32AccessDel(pszServername, pszResource);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg
- DosOpen

Related Information

For information about:

- Defining user name or group name access permissions, see [Access Permission - NetAccessAdd](#).
- Listing all permissions and resources, see [Access Permission - NetAccessEnum](#).

NetAccessEnum or Net32AccessEnum

NetAccessEnum or Net32AccessEnum

The NetAccessEnum API enumerates all access permission records.

Restrictions

LAN Server and OS/2 Warp Server

This API can be called from OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. OS/2 workstations can issue this call locally only if the Peer service is started. User authority is allowed limited access to this API. Administrator authority is required for full access.

Directory and Security Server Only

A DSS server always returns NERR_InvalidAPI.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessEnum(pszServername, pszResource,
              recursive, sLevel, buf, usBuflen,
              pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32AccessEnum(pszServername, pszResource,
                recursive, ulLevel, buf, ulBuflen,
                pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- recursive* (16-bit short) enables or disables recursive searching. If *recursive* is off (0), NetAccessEnum returns entries only for the resource named. If *recursive* is on (nonzero), NetAccessEnum also returns an entry for each resource that branches from *pszResource* and has a different profile than *pszResource*. (For example, when a subdirectory or file has a different access profile than its parent directory.)
- sLevel* or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Resource Access Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.

ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg
- DosOpen

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

The NetAccessEnum API can return entries only for an application having ACCESS_PERM permissions. If the user does not have administrative privileges, NetAccessEnum does not return an ERROR_ACCESS_DENIED error code. It returns NERR_Success error code with 0 entries.

The *pszResource* parameter limits the entries returned by NetAccessEnum. If *pszResource* is not a null string, it serves as a prefix for the path name. For example, if *pszResource* is C:\PROG, NetAccessEnum returns access permission records for resources that begin with C:\PROG.

The *pusEntriesAvail* parameter indicates the number of entries available for the given *pszResource* and *recursive* parameters, not the total number of entries in the access file.

Therefore, NetAccessEnum returns information only for resources without default settings below the root directory specified in the request. This is consistent semantically with a standard LAN Server API, which returns only explicit permissions. In addition, it is highly recommended that the recursive switch always be set to FALSE.

Related Information

For information about:

- Adding an access permission record, see [Access Permission - NetAccessAdd](#).
- Retrieving information about the permissions for a resource, see [Access Permission - NetAccessGetInfo](#).

NetAccessGetInfo or Net32AccessGetInfo

NetAccessGetInfo or Net32AccessGetInfo

The NetAccessGetInfo API retrieves information about the access permission record of a resource. This API returns the data structures described in [Resource Access Data Structures](#). It can return either an *access_info_0* data structure or an *access_info_1* data structure followed by 0 or more *access_list* data structures. The number of *access_list* data structures returned is specified in the *acc1_count* field of the *access_info_1* data structure.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. OS/2 workstations can issue this call locally only if the Peer service is started. User authority is allowed limited access to this API. Administrator authority is required for full access.

Directory and Security Server Only

A DSS server always returns NERR_InvalidAPI.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessGetInfo(pszServername, pszResource, sLevel, buf,
                usBuflen, pusBytesAvail);    /* 16 bit */

Net32AccessGetInfo(pszServername, pszResource, ulLevel, buf,
                  pulBytesAvail);           /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Resource Access Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.

ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg
- DosOpen

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

If the calling process does not have administrative privileges, NetAccessGetInfo can be called successfully only by a process that has special permissions (ACCESS_PERM) defined in the access permission record of the resource.

The specified *pszResource* must be a complete path name.

In specifying the *pszQueueName* for a resource, use the name originally assigned to the resource with the NetShareAdd API.

If *sLevel* is 1, NetAccessGetInfo returns an *access_info_1* data structure followed by an *access_list* data structure for each entry in the list of the resource. The number of entries can be determined by examining the *acc1_count* component in the *access_info_1* data structure.

If *buf* cannot hold all of the fixed-length and variable-length data (all *access_list* data structures), NetAccessGetInfo returns the NERR_BufTooSmall error code. Most GetInfo functions return the ERROR_MORE_DATA error code when more data is available.

You can determine the necessary buffer size by calling this API with *usBuflen* (or *ulBuflen*) set to 0. In this case, the

API returns the number of bytes available in the *pusBytesAvail* (or *pulBytesAvail*) value. Then you can call this API again and allocate a buffer size (*usBuffer* or *ulBuffer*) at least as large as the returned bytes available.

Related Information

For information about:

- Listing all resources and permissions, see [Access Permission - NetAccessEnum](#).
- Changing the current permissions for a resource, see [Access Permission - NetAccessSetInfo](#).

NetAccessGetUserPerms or Net32AccessGetUserPerms

NetAccessGetUserPerms or Net32AccessGetUserPerms

The NetAccessGetUserPerms API supplies a specified user's or group's permission to a resource. The resource can be a file, directory, drive, or logical resource and can be specified remotely by a universal naming convention (UNC) path as well as by a server name.

The permissions returned are based on the user's entry and the entry for any groups to which the user belongs. Priority always is given to the user's entry, if one exists.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. OS/2 workstations can issue this call locally only if the Peer service is started.

The NetAccessGetUserPerms API requires administrator authority, with the exception that users always are allowed to request their own permissions to any resource. In addition, a user with ACCESS_PERM permission (see the pPermission parameter description) to the resource can get the permissions for any user or group.

Directory and Security Server Only

The rules for checking differ if the user ID being checked is not the callers or the caller is not a member of the group being checked.

This API is supported with the following restrictions:

- Group name checked returns access permissions based on the group and group_obj ACL entries.
- If the user name being checked is not the same as the caller, the API returns access based on user, other_obj, and any-other ACL entries.
- Caller must have 't' permission on the resource unless the caller is a member of the group being checked or the user ID is the caller's.
- Local calls get ERROR_ACCESS_DENIED returned if the caller is not logged on.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessGetUserPerms(pszServername, pszUserOrGroupID,
                     pszResource, pPermission);    /* 16 bit */

Net32AccessGetUserPerms(pszServername, pszUserOrGroupID,
                       pszResource, pPermission); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pPermission (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to a bit mask specifying the access permission level granted to the user or group ID to which *pszUserOrGroupID* points.

The ACCESS.H header file defines the following possible values for the bit mask:

SYMBOLIC CONSTANT	VALUE	MEANING
ACCESS_NONE	0	No permission to access the resource.
ACCESS_READ	0x01	Permission to read data from the resource and, by default, run the resource.
ACCESS_WRITE	0x02	Permission to write data to the resource.
ACCESS_CREATE	0x04	Permission to create an instance of the resource (such as a file); the resource is written to the resource when it is being created.
ACCESS_EXEC	0x08	Permission to run the resource.
ACCESS_DELETE	0x10	Permission to delete the resource.
ACCESS_ATTRIB	0x20	Permission to change the attributes of the resource (such as the date the resource was last changed).
ACCESS_PERM	0x40	Permission to change the permissions (read, write, create, execute) assigned to a resource for the application.
ACCESS_ALL	0x7F	Permission to read, write, create, execute, or delete a resource; change attributes or permissions.
ACCESS_GROUP	0x8000	Permission for a particular user or group. If returned, indicates that the resource is a group.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead

NetAccessSetInfo or Net32AccessSetInfo

NetAccessSetInfo or Net32AccessSetInfo

The NetAccessSetInfo API changes an access permission record for a resource.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. OS/2 workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

A DSS server always returns NERR_InvalidAPI.

Syntax

```
#include <netcons.h>
#include <access.h>

NetAccessSetInfo(pszServername, pszResource, sLevel,
                 buf, usBuflen, parmnum);    /* 16 bit */

Net32AccessSetInfo(pszServername, pszResource, ulLevel,
                  buf, ulBuflen, parmnum);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1, which specifies the *access_info_1* data structure.

parmnum (16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a specific field in the data structure is to be passed. If *parmnum* is 0, the entire *access_info_1* data structure is passed, followed by 0 or more *access_list* data structures. Otherwise, *parmnum* can be set to 2 (ACCESS_ATTR_PARMNUM), which specifies that only the *acc1_attr* field of the *access_info_1* data structure is passed.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The <i>sLevel</i> parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.

NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_ACFTooManyLists	2230	Too many lists were specified.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosOpen
- DosQFileInfo
- DosRead
- DosWrite

Remarks

The specified resource must be a complete path name. For example, directory resources must include a drive letter.

A user or application that has ACCESS_PERM permission for a particular resource can change the access permission record for that resource and remove the permissions for that resource.

The *parmnum* parameter is used only to change the *acc1/attr* field in the *access_info_1* data structure. To change the user or group permissions through NetAccessSetInfo, call NetAccessGetInfo first. If the user or group list is not complete, any information that is not included is lost.

Related Information

For information about:

- Listing server resources and permissions, see [Access Permission - NetAccessEnum](#).
- Retrieving permissions of a resource, see [Access Permission - NetAccessGetInfo](#).

ent_acl_replace_apply

ent_acl_replace_apply

This remote procedure call (RPC) propagates an access control list (ACL) to the specified directory and all of the files contained within the directory. Also, *ent_acl_replace_apply()* optionally propagates the ACL to all subtrees contained beneath the specified directory.

Restrictions

This RPC does not apply an ACL record to any of the following DSS server system subdirectories:

- \BMLAN
- \BMLAN\DCDB and its subdirectories
- \BMLAN\NETPROG and its subdirectories
- \BMLAN\DOSLAN and its subdirectories

Syntax

```
#include <dsserr.h>
#include <apply.h>

ent_acl_replace_apply (handle, resource_name, sec_acl_type,
                      *acl_to_apply, tolerance,
                      recursion, max_errors,
                      error_buffer[ ],
                      *num_errors, *status);
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- handle* (handle_t) RPC binding handle. Usually obtained by a call to *rpc_ns_binding_import_next()*.
- resource_name* (unsigned_char_p_t) points to a NULL terminated string specifying the name of the resource to start the apply operation from. The resource itself is included in the apply operation. The resource must be a directory.
- sec_acl_type* (sec_acl_type_t) specifies the ACL type. Container objects (directories) have three ACLs associated with them:

SYMBOLIC CONSTANT	MEANING
sec_acl_type_object	The ACL on the container itself.
sec_acl_type_default_objec	The initial ACL for any files cre container.
sec_acl_type_default_conta n	The initial ACL for directories c the container.

- acl_to_apply* (sec_acl_list_t) the ACL to apply. The original ACL is replaced with this ACL.
- tolerance* (32-bit unsigned) specifies how to handle files and directories that the caller does not have permission to change.

SYMBOLIC CONSTANT	MEANING
apply_tolerance_low	Halt the apply operation on the f DSS_AccessDenied error.
apply_tolerance_med	If a DSS_AccessDenied error occur error in the error_buffer and con essing.
apply_tolerance_high	Objects that the caller does not h

to change are skipped. No DSS_Acc errors are returned.

<i>recursion</i>	(32-bit unsigned) If TRUE, propagate the ACL down the entire tree. If FALSE, only <i>resource_name</i> and the files within <i>resource_name</i> will be affected.
<i>max_errors</i>	(32-bit unsigned) Maximum number of errors that can be returned in <i>error_buffer</i> array.
<i>error_buffer</i>	(error_entry_t) buffer used to return errors during apply operation. The client is responsible for calling <code>rpc_sm_client_free()</code> for each error returned to release storage used by <i>resource_name</i> .
<i>num_errors</i>	(32-bit unsigned) Number of errors returned in <i>error_buffer</i> array.
<i>status</i>	(error_status_t) A pointer to the status of the call.

The *acl_to_apply* must contain at least one entry that specifies the **c** permission bit.

Note: You must use the IDL Compiler to compile `apply.idl` to generate the client-side stub for the RPC call. Then compile the stub and link to the `.obj` file to resolve the external references.

ent_acl_manipulate_apply

ent_acl_manipulate_apply

This remote procedure call (RPC) propagates changes to ACLs based on a list of operations passed in by the caller. The operations add, update, and delete entries contained in each ACL. Change list processing is performed on the specified directory and all files contained within the directory. Also, *ent_acl_manipulate_apply()* optionally applies the process to all subtrees contained beneath the specified directory.

Restrictions

This RPC does not apply an ACL record to any of the following DSS server system subdirectories:

- \\BMLAN
- \\BMLAN\\DCDB and its subdirectories
- \\BMLAN\\NETPROG and its subdirectories
- \\BMLAN\\DOSLAN and its subdirectories

Syntax

```
#include <dsserr.h>
#include <apply.h>

ent_acl_manipulate_apply(handle, resource_name, sec_acl_type,
                        change_list, tolerance,
                        recursion, max_errors,
                        error_buffer[ ],
                        num_errors, *status);
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>handle</i>	(handle_t) RPC binding handle. Usually obtained by a call to <i>rpc_ns_binding_import_next()</i> .
<i>resource_name</i>	(unsigned_char_p_t) points to a NULL terminated string specifying the name of the resource from where to start the apply operation. The resource itself is included in the apply operation.

	The resource must be a directory.	
<i>sec_acl_type</i>	(<i>sec_acl_type_t</i>) Specifies the ACL type. Container objects (directories) have three ACLs associated with them:	
	SYMBOLIC CONSTANT	MEANING
	<i>sec_acl_type_object</i>	The ACL on the directory itself.
	<i>sec_acl_type_default_objec</i>	The initial ACL for any files cre directory.
	<i>sec_acl_type_default_conta</i>	nThe initial ACL for directories c the container.
<i>change_list</i>	Points to <i>ent_entries_to_manipulate_t</i> type which essentially is a list of <i>sec_acl_entry_t</i> with associated actions.	
<i>tolerance</i>	(32-bit unsigned) specifies how to handle files and directories that the caller does not have permission to change.	
	SYMBOLIC CONSTANT	MEANING
	<i>apply_tolerance_low</i>	Halt the apply operation on the f DSS_AccessDenied error.
	<i>apply_tolerance_med</i>	If a DSS_AccessDenied error occur error in the error_buffer and con essing.
	<i>apply_tolerance_high</i>	Objects that the caller does not h to change are skipped. No DSS_Acc errors are returned.
<i>recursion</i>	(32-bit unsigned) If TRUE, propagates the changes down the entire tree. If FALSE, only <i>resource_name</i> and the files contained within <i>resource_name</i> will be affected.	
<i>max_errors</i>	(32-bit unsigned) Maximum number of errors that can be returned in <i>error_buffer</i> array.	
<i>error_buffer</i>	(<i>error_entry_t</i>) Buffer used to return errors during apply operation. The client is responsible for calling <i>rpc_sm_client_free()</i> for each error returned to release storage used by <i>resource_name</i>	
<i>num_errors</i>	(32-bit unsigned) Number of errors returned in <i>error_buffer</i> array.	
<i>status</i>	(<i>error_status_t</i>) A pointer to the status of the call. For each object processed, the ACL that results from the manipulations listed in <i>change_list</i> must have at least one entry that contains the c permission bit or the change fails for that object.	

Note: You must use the IDL Compiler to compile *apply.idl* to generate the client-side stub for the RPC call. Then compile the stub and link to the .obj file to resolve the external references.

Alert Category

This category includes the following APIs:

Alert - NetAlertRaise
Alert - NetAlertStart
Alert - NetAlertStop

Alert APIs provide a system for notifying network service programs and applications of network events. They are used with the ALERT.H and NETCONS.H header files.

An *event* is a particular instance of a process or state of hardware defined by either an application or the LAN Server software. LAN Server sends out an *alert*, in the form of a message or the resetting of a semaphore, when certain events occur. Other programs, network services, or internal network components use the NetAlertRaise API to raise an alert, notifying various applications or users when a particular type of event occurs.

The ALERT.H header file defines the following classes of events for the alerts that are sent out:

- A print job was completed.
- A user or application received a broadcast message.
- An entry was added to an error log file.
- A network event required administrative assistance.
- A user accessed or used certain applications or resources.

Other classes of alerts can be defined for network applications as needed. For example, if an application routinely writes large amounts of data to a disk drive, running the risk of filling the disk, the user might want the event of no free disk space to trigger an alert that notifies the application to pause or end the process slowing the system.

An application or network service, known as a *client*, registers to be notified of an event (or class of events) by calling the NetAlertStart API. Each registration adds an entry to an alert table.

A client can receive alert messages through one of two delivery mechanisms:

- A mailslot (registered as *\mailslot\name*).
- A system semaphore (registered as *lsem\name*). If a program requires detailed information about an event, it should be registered as a mailslot, because a semaphore cannot transmit such information.

A client can be registered for one type of event, or it can be registered for several types by calling the NetAlertStart API a number of times.

To discontinue alerts for a registered client, use the NetAlertStop API to remove that entry in the alert table for the particular class of event.

To change the internal size of the alert table (thus allowing more alerts to be defined), a user or application must change the *numalert* component in the IBMLAN.INI file, and then restart the requester (by using the NET STOP RDR /Y and NET START RDR commands).

Alert Data Structures

An application registered as a mailslot client receives information about each class of event for which it is registered. This information consists of a fixed-length header followed by variable-length information specific to the type of event, as defined in the ALERT.H header file.

Alert Structure Header

The fixed-length header contains the following data:

```
/* Standard event data structure */

struct std_alert {
    long          alrt_timestamp;
    unsigned char alrt_eventname[EVLLEN+1];
    unsigned char alrt_pad1;
    unsigned char alrt_servicename[SNLEN+1];
};
```

where:

- *alrt_timestamp* indicates the time and date of the event.
- *alrt_eventname* is an ASCII string indicating the alert class (type of event).
- *alrt_pad1* word-aligns data structure components.
- *alrt_servicename* is an ASCII string indicating the application that is raising the alert.

Event Structures

The ALERT.H header file contains data structures for predefined alert classes. These structures define only the fixed-length part of the information, not the ASCIIZ strings that follow some of the structures.

There are five alert types, but only four of these are actual data structures. These data structures are arrays, represented by square brackets ([]). The Network Message Received alert is not a data structure because it consists only of the ASCIIZ string of message text received. Each of the five alert types is described in the following sections.

Print Request Completed

```
struct print_other_info {
    short alrtpr_jobid;
    short alrtpr_status;
    long  alrtpr_submitted;
    long  alrtpr_size;
};
/* followed by consecutive ASCIIZ strings

unsigned char computername[];
unsigned char username[];
unsigned char queueename[];
unsigned char destname[];
unsigned char status_string[];

*/
```

where:

- *alrtpr_jobid* is the identification number of the print job.
 - *alrtpr_status* indicates the status of the print job.
 - *alrtpr_submitted* is a time stamp indicating when the print job was submitted.
 - *alrtpr_size* indicates the size (in bytes) of the print job.
 - *computername* is an ASCIIZ string indicating the requester or server that submitted the print job.
 - *username* is an ASCIIZ string indicating the user that requested the printing.
 - *queueename* is an ASCIIZ string indicating the queue that handled the print job.
 - *destname* is an ASCIIZ string indicating the printer that handled the job.
 - *status_string* is information that the print processor returns. This string corresponds to *status_string* in the *printjob* data structure for the print job.
-

Network Message Received

In this case, no data structure is defined; however, the text from the received message is in the following format:


```
char msg_text [];
```

where *msg_text* is an ASCIIZ string of message text.

Entry Made to Error Log File

```
struct errlog_other_info {
    short alrter_errcode;
    long alrter_offset;
};
```

where:

- *alrter_errcode* is the error code that was logged.
 - *alrter_offset* is the offset for the new entry in the error log file.
-

Notify Administrator of Network Event

```
struct admin_other_info {
    short alrtad_errcode;
    short alrtad_numstrings;
};

/* followed by 0-9 consecutive ASCIIZ strings
unsigned char mergestrings [][];

*/
```

where:

- *alrtad_errcode* is the error code for the new message in the message log file.
 - *alrtad_numstrings* indicates the number (0-9) of consecutive ASCIIZ strings that *mergestrings* contains.
 - *mergestrings* is a series of consecutive ASCIIZ strings that make up the error message indicated by *alrtad_errcode*.
-

Notify User of an Event

```
struct user_other_info {
    short alrtus_errcode;
    short alrtus_numstrings;
};

/* followed by the consecutive ASCIIZ strings
unsigned char mergestrings [][];
```

```

unsigned char username[];
unsigned char computername[];

* /

```

where:

- *alrtus_errcode* is the error code for the new message in the message log file.
- *alrtus_numstrings* indicates the number (0-9) of consecutive ASCIIZ strings that *mergestrings* contains.
- *mergestrings* is a series of consecutive ASCIIZ strings that make up the error message indicated by *alrtus_errcode*.
- *username* is the name of the user or application affected by the alert.
- *computername* is the name of the computer that the user or application is accessing.

The ALERT.H header file contains macros to simplify access to the variable-length fields in the *alert* structure as follows:

MACRO	TASK
ALERT_OTHER_INFO, ALERT_OTHER_INFO_F	When given a pointer to the start of the std_alert data structure, the ALERT_OTHER_INFO macro resolves to a pointer to the variable-length part of the alert message (the information specific to the alert class). When a far pointer is required, use ALERT_OTHER_INFO_F.
ALERT_VAR_DATA, ALERT_VAR_DATA_F	Works with the data structures defined in the ALERT.H header file. Given a pointer to the beginning address of the data structure, ALERT_VAR_DATA returns a pointer to the first variable-length ASCIIZ string. When a far pointer is required, use ALERT_VAR_DATA_F.

Related Information

For information about creating mailslots, see [Mailslot Category](#).

NetAlertRaise or Net32AlertRaise

NetAlertRaise or Net32AlertRaise

The NetAlertRaise API notifies all clients registered as semaphores or mailslots in the alert table that a particular event has occurred. Semaphores are cleared, and mailslots are sent messages.

For mailslot clients, NetAlertRaise writes information from *buf* to clients registered as mailslots by calling the DosWriteMailslot API.

All semaphore clients must be created with the NoExclusive option set and must be called by a process that calls the DosMuxSemWait API on the semaphore. This procedure informs the process of the state transition of the semaphore.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

Syntax

```
#include <netcons.h>
#include <alert.h>

NetAlertRaise(pAlertType, buf,
              usBuflen, timeout);    /* 16 bit */

Net32AlertRaise(pAlertType, buf,
                ulBuflen, timeout); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pAlertType</i>	(const unsigned char LSFAR *) points to an ASCIIZ string indicating the type of alert to raise. The ALERT.H header file defines the following types of alerts:		
	SYMBOLIC CONSTANT	ASCIIZ STRING	MEANING
	ALERT_ADMIN_EVENT	"ADMIN"	Notify an administrator
	ALERT_ERRORLOG_EVENT	"ERRORLOG"	Entry added to error log
	ALERT_MESSAGE_EVENT	"MESSAGE"	User or application request message.
	ALERT_PRINT_EVENT	"PRINTING"	Print job completed or pending
	ALERT_USER_EVENT	"USER"	Application or resource change
	NOTE: Other types of events can be defined as necessary.		
<i>buf</i>	(const unsigned char LSFAR *) points to the data structures defined in Alert Data Structures .		
<i>timeout</i>	(unsigned long) specifies the number of milliseconds to wait for event information to be written to the mailslot. A value of -1 means wait forever.		

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_NoSuchAlert	2432	An incorrect or nonexistent alert name

was raised.

Other codes could be returned from the following functions:

- DosFsRamSemRequest
- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about:

- Creating a mailslot, see [Mailslot - DosMakeMailslot](#).
- Registering a client for an event, see [Alert - NetAlertStart](#).
- Ending event watching, see [Alert - NetAlertStop](#).

NetAlertStart or Net32AlertStart

NetAlertStart or Net32AlertStart

The NetAlertStart API registers a client to be notified of a particular type of network event.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

Syntax

```
#include <netcons.h>
#include <alert.h>

NetAlertStart(pAlertType, pszRecipient, maxdata);      /* 16 bit */

Net32AlertStart(pAlertType, pszRecipient, maxdata);   /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pAlertType</i>	(const unsigned char LSFAR *) points to an ASCIIZ string indicating the type of alert to raise. The ALERT.H header file defines the following types of alerts:		
	SYMBOLIC CONSTANT	ASCIIZ STRING	MEANING
	ALERT_ADMIN_EVENT	"ADMIN"	Notify an administrator
	ALERT_ERRORLOG_EVENT	"ERRORLOG"	Entry added to error log
	ALERT_MESSAGE_EVENT	"MESSAGE"	User or application received message.
	ALERT_PRINT_EVENT	"PRINTING"	Print job completed or pending
	ALERT_USER_EVENT	"USER"	Application or resource accessed
	NOTE: Other types of events can be defined as necessary.		

pszRecipient (const unsigned char LSFAR *) points to an ASCIIZ string specifying the mailslot or semaphore client to receive the alerts.

maxdata (16-bit unsigned short or 32-bit unsigned long) specifies a limit (in bytes) to the information the mailslot client can receive about events of a specified type.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BadEventName	2143	The event name is incorrectly formed.
NERR_AlertExists	2430	The specified client is already registered for the specified event.
NERR_TooManyAlerts	2431	The Alerter service table is full.
NERR_BadRecipient	2433	The Alerter service recipient is not valid.

Other codes could be returned from the following functions:

- DosFsRamSemRequest
- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Remarks

Event names are ASCIIZ strings stored in the ALERT.H header file. Applications can define their own events, specifying an event name when calling the NetAlertStart and NetAlertRaise APIs. If you create an event data structure, choose a name that does not duplicate a name used by another application.

If pszRecipient is a semaphore, calling NetAlertRaise for the specified event opens, clears, resets, and closes the system semaphore. The process owning the semaphore must have created it with the NoExclusive option set. Presumably such a process is running a DosSemWait or DosMuxSemWait function on the semaphore and has the transition status of the semaphore.

If NetAlertStart starts a particular alert, the alert still exists (even when a process is ended) until the NetAlertStop API is called to stop the alert.

Related Information

For information about:

- Creating a mailslot, see [Mailslot - DosMakeMailslot](#).
- Creating a semaphore, see `DosCreateSem` in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Reading a mailslot, see [Mailslot - DosReadMailslot](#).
- Ending the watch of a client for a class or type of event, see [Alert - NetAlertStop](#).

NetAlertStop or Net32AlertStop

NetAlertStop or Net32AlertStop

The `NetAlertStop` API removes a registered client from the alert table.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

Syntax

```
#include <netcons.h>
#include <alert.h>

NetAlertStop(pAlertType, pszRecipient);      /* 16 bit */
Net32AlertStop(pAlertType, pszRecipient);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pAlertType</i>	(const unsigned char LSFAR *) points to an ASCIIZ string indicating the type of alert to raise. The ALERT.H header file defines the following types of alerts:		
	SYMBOLIC CONSTANT	ASCIIZ STRING	MEANING
	ALERT_ADMIN_EVENT	"ADMIN"	Notify an administrator
	ALERT_ERRORLOG_EVENT	"ERRORLOG"	Entry added to error log
	ALERT_MESSAGE_EVENT	"MESSAGE"	User or application received message.
	ALERT_PRINT_EVENT	"PRINTING"	Print job completed or pending
	ALERT_USER_EVENT	"USER"	Application or resource used
	NOTE: Other types of events can be defined as necessary.		
<i>pszRecipient</i>	(const unsigned char LSFAR *) points to an ASCIIZ string specifying the mailslot or semaphore client to receive the alerts.		

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_NoSuchAlert	2432	An incorrect or nonexistent alert name was raised.

Other codes could be returned from the following functions:

- DosFsRamSemRequest
- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about registering a client to watch for a particular event, see [Alert - NetAlertStart](#).

Alias Category

This category includes the following APIs:

[Alias - NetAliasAdd](#)
[Alias - NetAliasDel](#)
[Alias - NetAliasEnum](#)
[Alias - NetAliasGetInfo](#)
[Alias - NetAliasSetInfo](#)

Alias APIs examine or change aliases. In OS/2 LAN Server and in DSS, an *alias* is a nickname for a server resource. An *alias type* can be either file, print, or serial. They are used with both the NETCONS.H and DCDB.H header files.

Alias Data Structures

Alias Levels 0, 1, and 2 data structures apply generally. Alias Levels 10, 11, and 12 data structures apply only to DSS.

Alias Level 0

Alias level 0 is structured as follows.

```
struct alias_info_0 {
    unsigned char    ai0_alias[ALIAS_LEN + 1];
};
```

where *ai0_alias* is the name by which the resource is known within a domain or resource domain. An alias must be unique within the domain or resource domain.

Alias Level 1

```
struct alias_info_1 {
    unsigned char    ail_alias[ALIAS_LEN + 1];
    unsigned char    ail_pad;
    unsigned char    LSFAR * LSPTR ail_remark;
    unsigned short   ail_type;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *ail_pad* word-aligns the data structure component.
- *ail_remark* points to an ASCIIZ string containing an optional comment about the alias. The string must be no longer than 40 bytes. A null string indicates no remark.
- *ail_type* is one of three values indicating the alias type. The following values are defined in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
ALIAS_TYPE_FILE	0x0001	File alias
ALIAS_TYPE_PRINTER	0x0002	Printer alias
ALIAS_TYPE_SERIAL	0x0004	Serial device alias

Alias Level 2

```
struct alias_info_2 {
    unsigned char    ai2_alias[ALIAS_LEN + 1];
    unsigned char    ai2_pad_1;
    unsigned char    LSFAR * LSPTR ail_remark;
    unsigned short   ai2_type;
    unsigned short   ai2_location;
    unsigned char    ai2_server[CNLEN + 1];
    unsigned short   ai2_mode;
    unsigned short   ai2_maxuses;
    unsigned char    ai2_netname[NNLEN + 1];
    unsigned char    ai2_pad_2;
    unsigned char    ai2_queue[QNLEN + 1];
    unsigned char    ai2_pad_3;
};
```



```

unsigned char LSFAR * LSPTR    ai2_path;
unsigned short                                ai2_priority;
unsigned char LSFAR * LSPTR    ai2_device_pool;
};

```

where:

- The first field in this data structure is identical to that in the previous level.
- *ai2_pad_1* word-aligns the data structure component.
- The next two fields are identical to the same fields in the previous level.
- *ai2_location* is reserved and must be 0.
- *ai2_server* is the name of the server where the resource described by this alias resides.
- *ai2_mode* is one of three values that indicate when the alias is shared. The ALIAS_MODE_DYNAMIC Symbolic Constant applies to file aliases only. The DCDB.H header file defines these values as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
ALIAS_MODE_STARTUP	0x0000	The resource is shared when the server is started.
ALIAS_MODE_BYADMIN	0x0001	The resource must be explicitly shared by the administrator.
ALIAS_MODE_DYNAMIC	0x0002	The resource is shared when a request to assign it is made.

- *ai2_maxuses* indicates the maximum number of users who can have redirection to the resource identified by this alias. Values from 0-65535 are valid.
- *ai2_netname* is the alias name for file aliases, and the queue name for printer and serial device aliases. For NetAliasAdd and NetAliasSetInfo operations, any data in this field is ignored.
- *ai2_pad_2* word-aligns the data structure component.
- *ai2_queue* applies to printer and serial device aliases only. For serial devices, the queue is identical to the alias. For printers, the queue is the print destination.
- *ai2_pad_3* word-aligns the data structure component.
- *ai2_path* applies to file aliases only. It points to an ASCIIZ string indicating a path that is relative to the server on which the alias will be shared. The path must consist of a drive letter, a colon (:) and a backslash (\); for example C:\. The maximum length of the string is 254 bytes.
- *ai2_priority* applies to serial device aliases only. The priority can have a value from 1-9, with 1 being the highest priority.
- *ai2_device_pool* applies to serial device aliases only and must not be NULL. It points to an ASCIIZ list of serial devices separated by blanks. Valid serial devices are LPT1-LPT9 and COM1-COM16.

Alias Level 10

This data structure applies only to DSS.

```

struct alias_info_10 {
    unsigned char LSFAR * LSPTR    ai10_alias;
    unsigned char LSFAR * LSPTR    ai10_global_name;
    unsigned char                                ai10_pad;
};

```

where:

- *ai10_alias* is the name by which the resource is known within a resource domain. An alias must be unique within the resource domain.
- *ai10_global_name* is a name that is universally meaningful and usable from anywhere in the DCE naming environment. The prefix a /... or a /.: indicates that a name is global.
- *ai10_pad* word-aligns the data structure component.

Alias Level 11

This data structure applies only to DSS.

```
struct alias_info_11 {
    unsigned char LSFAR * LSPTR          ai11_alias;
    unsigned char LSFAR * LSPTR          ai11_global_name;
    unsigned char                                     ai11_pad[2 ];
    unsigned char LSFAR * LSPTR          ai11_remark;
    unsigned short                                     ai11_type;
};
```

where:

- The first two fields in this data structure are identical to those in the previous level.
- *ai11_pad* word-aligns the data structure component.
- *ai11_remark* points to an ASCIIZ string containing an optional comment about the alias. The string must be no longer than 40 bytes. A null string indicates no remark.
- *ai11_type* is one of three values indicating the alias type. The following values are defined in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
ALIAS_TYPE_FILE	0x0001	File alias
ALIAS_TYPE_PRINTER	0x0002	Printer alias
ALIAS_TYPE_SERIAL	0x0004	Serial device alias

Alias Level 12

This data structure applies only to DSS.

```
struct alias_info_12 {
    unsigned char LSPTR * LSPTR          ai12_alias;
    unsigned char LSPTR * LSPTR          ai12_global_name;
    unsigned char                                     ai12_pad_1 [2];
    unsigned char LSFAR * LSPTR          ai12_remark;
    unsigned short                                     ai12_type;
    unsigned short                                     ai12_location;
    unsigned char LSPTR * LSPTR          ai12_server;
    unsigned char                                     ai12_pad_4[12 ];
    unsigned short                                     ai12_mode;
    unsigned short                                     ai12_maxuses;
};
```

```

unsigned char  LSPTR * LSPTR          ai12_netname;
unsigned char  ai12_pad_2[10];
unsigned char  ai12_queue[QNLEN + 1];
unsigned char  ai12_pad_3;
unsigned char  LSFAR * LSPTR          ai12_path;
unsigned short ai12_priority;
unsigned char  LSFAR * LSPTR          ai12_device_pool;
};

```

where:

- The first 5 fields in this data structure are identical to those in the previous level.
- *ai12_pad_1* aligns the data structure component.
- The next two fields are identical to the same fields in the previous level.
- *ai12_location* is reserved and must be 0.
- *ai12_server* is the name of the server where the resource described by this alias resides.
- *ai12_mode* is one of three values that indicate when the alias is shared. The ALIAS_MODE_DYNAMIC Symbolic Constant applies to file aliases only. The DCDB.H header file defines these values as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
ALIAS_MODE_STARTUP	0x0000	The resource is shared when the server is started.
ALIAS_MODE_BYADMIN	0x0001	The resource must be explicitly shared by the administrator.
ALIAS_MODE_DYNAMIC	0x0002	The resource is shared when a request to assign it is made.

- *ai12_maxuses* indicates the maximum number of users who can have redirection to the resource identified by this alias. Values from 0-65535 are valid.
- *ai12_netname* is the alias name for file aliases, and the queue name for printer and serial device aliases. For NetAliasAdd and NetAliasSetInfo operations, any data in this field is ignored.
- *ai12_pad_2* word-aligns the data structure component.
- *ai12_queue* applies to printer and serial device aliases only. For serial devices, the queue is identical to the alias. For printers, the queue is the print destination.
- *ai12_pad_3* word-aligns the data structure component.
- *ai12_path* applies to file aliases only. It points to an ASCIIZ string indicating a path that is relative to the server on which the alias will be shared. The path must consist of a drive letter, a colon (:) and a backslash (\); for example C:\. The maximum length of the string is 254 bytes.
- *ai12_priority* applies to serial device aliases only. The priority can have a value from 1-9, with 1 being the highest priority.
- *ai12_device_pool* applies to serial device aliases only and must not be NULL. It points to an ASCIIZ list of serial devices separated by blanks. Valid serial devices are LPT1-LPT9 and COM1-COM16.

NetAliasAdd or Net32AliasAdd

NetAliasAdd or Net32AliasAdd

The NetAliasAdd API creates an alias definition.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAliasAdd requires one of the following:

- The calling process has Administrator authority.
- The calling process has server operator privilege.
- If you are adding a print alias, the calling process has print operator privilege.
- If you are adding a serial device alias, the calling process has comm operator privilege.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAliasAdd(pszTarget, sLevel,
           buf, usBuflen);          /* 16 bit */

Net32AliasAdd(pszTarget, ulLevel,
             buf, ulBuflen, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.
<i>null</i> or <i>lservername</i>	added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomname</i>	added to resource domain of the local cell when connected to a DSS cell.
<i>//resdomname@cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 2 for OS/2 LAN Server, and OS/2 Warp Server, which specifies the data structure described in Alias Level 2 .
	For DSS, see Alias Level 2 or Alias Level 12.
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size <code>lsdce_err_stat_t</code> (defined in <code>lsdceerr.h</code>). Contains specific DCE error information when <code>NERR_DCEError</code> is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_DATA	13	The specified data is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_BAD_PATHNAME	161	Path name not valid.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_AliasExists	2782	The alias to be added already exists.
NERR_DCDLError	2786	A domain control database is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddress

Remarks

An alias that is created with an *ai2_mode* value of ALIAS_MODE_STARTUP is not shared by the NetAliasAdd API. Refer to [Share - NetShareAdd](#) for information about sharing resources.

Related Information

For information about:

- Listing all alias definitions, see [Alias - NetAliasEnum](#).
- Changing an alias definition, see [Alias - NetAliasSetInfo](#).
- Removing an alias definition, see [Alias - NetAliasDel](#).
- Automatically assigning aliases to a user at logon time, see [User - NetUserGetLogonAsn](#) and [User - NetUserSetLogonAsn](#).

NetAliasDel or Net32AliasDel

NetAliasDel or Net32AliasDel

The NetAliasDel API deletes an alias definition.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAliasDel requires one of the following:

- The calling process has Administrator authority.
- The calling process has server operator privilege.
- If you are deleting a printer alias, the calling process has print operator privilege.
- If you are deleting a serial device alias, the calling process has comm operator privilege.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAliasDel(pszTarget pszAlias, reserved);      /* 16 bit */

Net32AliasDel(pszTarget, pszAlias, reserved,
             pStatusbuf);                      /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszTarget

is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.

null or lservername

added to DCDB of specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.

//resdomname

added to

		resource domain of the local cell when connected to a DSS cell.
	<i>//resdomainname@cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>pszAlias</i>	(unsigned char LSFAR *) points to an ASCIIZ string specifying the alias to be acted upon. For DSS, this may be a global name.	
<i>reserved</i>	(unsigned long) is reserved and must be 0.	
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size lsdce_err_stat_t (defined in lsdceerr.h). Contains specific DCE error information when NERR_DCEError is returned.	

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_AliasNotFound	2783	The alias does not exist.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.

NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Related Information

For information about:

- Creating an alias definition, see [Alias - NetAliasAdd](#).
- Listing all alias definitions, see [Alias - NetAliasEnum](#).
- Changing an alias definition, see [Alias - NetAliasSetInfo](#).

NetAliasEnum or Net32AliasEnum

NetAliasEnum or Net32AliasEnum

The NetAliasEnum API returns information about all aliases of a given type.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAliasEnum(pszTarget, sLevel, type, buf, usBuflen,
             pusEntriesReturned, pusEntriesAvail);          /* 16 bit */

Net32AliasEnum(pszTarget, ulLevel, type, buf, ulBuflen,
               pulEntriesReturned, pulEntriesAvail, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.
<i>null or llservname</i>	added to DCDB of a specified server when

		connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
	<i>//resdomname</i>	added to resource domain of the local cell when connected to a DSS cell.
	<i>//resdomname @cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0, 1, or 2 for OS/2 LAN Server and OS/2 Warp Server request, specifying which data structure to use, as described in Alias Data Structures . For DSS see Alias Level 10 , Alias Level 11 , and Alias Level 12.	
<i>type</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the kind of alias to be enumerated, as defined in DCDB.H:	
	SYMBOLIC CONSTANT	BIT MASK
	ALIAS_TYPE_FILE	0x0001
	ALIAS_TYPE_PRINTER	0x0002
	ALIAS_TYPE_SERIAL	0x0004
	MEANING	File alias Printer alias Serial device
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size <code>lsdce_err_stat_t</code> (defined in <code>lsdceerr.h</code>). Contains specific DCE error information when <code>NERR_DCEError</code> is returned.	

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The <code>sLevel</code> parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.

NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Retrieving information for a particular alias definition, see [User - NetUserGetLogonAsn](#).
- Listing aliases assigned to a particular user at logon time, see [Alias - NetAliasGetInfo](#).

NetAliasGetInfo or Net32AliasGetInfo

NetAliasGetInfo or Net32AliasGetInfo

The NetAliasGetInfo API retrieves information about the specified alias.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAliasGetInfo(pszTarget, pszAlias, usLevel, buf,
               usBuflen, pusBytesAvail);    /* 16 bit */

Net32AliasGetInfo(pszTarget, pszAlias, ulLevel, buf,
                 ulBuflen, pulBytesAvail, pStatusbuf);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.
<i> null or llservname</i>	added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i> //resdomname</i>	added to resource domain of the local cell when connected to a DSS cell.
<i> //resdomname@cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>pszAlias</i>	(unsigned char LSFAR *) points to an ASCIIZ string indicating the alias to be acted upon. For DSS, this may be a global name.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0, 1, or 2 for OS/2 LAN Server and OS/2 Warp Server request, specifying which data structure to use, as described in Alias Data Structures . For DSS see Alias Level 10 , Alias Level 11 , and Alias Level 12.
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size lsdce_err_stat_t (defined in lsdceerr.h). Contains specific DCE error information when NERR_DCEError is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.

ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_AliasNotFound	2783	The alias does not exist.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_DCEError	7500	DCE error status returned.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Changing an alias definition, see [Alias - NetAliasSetInfo](#).
- Listing alias definitions, see [Alias - NetAliasEnum](#).

NetAliasSetInfo or Net32AliasSetInfo

NetAliasSetInfo or Net32AliasSetInfo

The NetAliasSetInfo API changes information for an alias.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

At Level 1, process with any privilege level that successfully can run a call to NetAliasSetInfo. At Level 2, a successful call to NetAliasSetInfo requires one of the following:

- The calling process has administrator authority.
- The calling process has server operator privilege.
- If you are setting information for a print alias, the calling process has print operator privilege.
- If you are setting information for a serial device alias, the calling process has comm operator privilege.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAliasSetInfo(pszTarget, pszAlias, usLevel, buf,
               usBuflen, parmnum);          /* 16 bit */

Net32AliasSetInfo(pszTarget, pszAlias, ulLevel, buf,
                 ulBuflen, parmnum, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.	
<i>null or lservername</i>		added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomname</i>		added to resource domain of the local cell when connected to a DSS cell.
<i>//resdomname@cellname</i>		added to the resource domain of the specified cellname when

connected to a DSS cell.

<i>pszAlias</i>	(unsigned char LSFAR *) points to an ASCIIZ string that specifies the alias to be acted upon. For DSS, may be a global name.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 1 or 2 for OS/2 LAN Server and OS/2 Warp Server request, specifying which data structure to use, as described in Alias Level 1 and Alias Level 2 . For DSS see Alias Level 11 or Alias Level 12.
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size <code>lsdce_err_stat_t</code> (defined in <code>lsdceerr.h</code>). Contains specific DCE error information when <code>NERR_DCEError</code> is returned.
<i>parmnum</i>	(16-bit unsigned short or 32-bit unsigned long) is an integer that specifies whether the entire data structure or only a specific field in the data structure is to be passed. If the value is 0, the entire data structure is sent. Otherwise, <i>parmnum</i> specifies which field in the data structure is to be sent. For this API, <i>parmnum</i> can be any one of the following:

SYMBOLIC CONSTANT	VALUE	COMPONENT	COMMENT
ALIAS_REMARK_PARMNUM	3	aiX_remark	-
ALIAS_SERVER_PARMNUM	6	aiY_server	-
ALIAS_MODE_PARMNUM	7	aiY_mode	-
ALIAS_MAXUSES_PARMNUM	8	aiY_maxuses	-
ALIAS_PATH_PARMNUM	13	aiY_path	Valid for file aliases only
ALIAS_PRIORITY_PARMNUM	14	aiY_priority	Valid for serial device aliases only
ALIAS_DEVICEPOOL_PARMNUM	15	aiY_device_po	Valid for serial device aliases only

NOTE: X = 1 or 2. Y = 2.

If you specify *parmnum* to be 0, you must allocate enough memory space (specified by *buf* and *usBuflen*) to contain the entire data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.

ERROR_INVALID_DATA	13	The specified data is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_BAD_PATHNAME	161	Path name not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_AliasNotFound	2783	The alias does not exist.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddress

Related Information

For information about retrieving information about a particular alias definition, see [Alias - NetAliasGetInfo](#).

Application Category

This category includes the following APIs:

[Application - NetAppAdd](#)
[Application - NetAppDel](#)
[Application - NetAppEnum](#)
[Application - NetAppGetInfo](#)
[Application - NetAppSetInfo](#)

Application APIs manage information about network applications. They are used with the NETCONS.H and DCDB.H header files.

There are two types of data associated with applications: fixed data and list data. *Fixed data* includes the name of the application, the command line used to invoke the application, its location (local or remote, alias or drive, and path), name of a working directory (if any) associated with the application, and so on.

List data consists of a set of 0 or more structures that the application requires. For example, an application might require a resource containing certain data files. Redirections to these resources are made when the application is invoked. The redirections are deleted when the application stops.

An application always has fixed data associated with it; it might or might not have list data associated with it.

There are four application types:

- Public DOS applications
- Private DOS applications
- Public OS/2 applications
- Private OS/2 applications

Program locations can be local or remote. If the location is local, the executable file used to start the application must be installed on each requester where it will run. If the location is remote, the application is stored on a server.

Application Information Data Structures

Application APIs for OS/2 LAN Server and OS/2 Warp Server use only Levels 0 through Level 3. DSS Application APIs use all the levels, Level 0 through Level 13.

Application Information Level 0

```
struct app_info_0 {
    unsigned char    app0_name[APP_LEN + 1];
};
```

where:

- *app0_name* is an ASCIIZ string that specifies an identifying name for the application. You can use any characters except embedded blanks and the following:

" / \ [] : | < > + = ; , . ? *

Application Information Level 1

```
struct app_info_1 {
    unsigned char    appl_name[APP_LEN + 1];
    unsigned char    appl_pad_1;
    unsigned char LSFAR * LSPTR    appl_remark;
};
```


where:

- The first field in this data structure is identical to that in the previous level.
- *app1_pad_1* word-aligns the data structure component.
- *app1_remark* points to an ASCIIZ string that provides a description of the application. The string must not be longer than 40 bytes. The string appears as the application's title in the applications folder. Null strings are not allowed; a remark is required. The remark string cannot include a backslash (\) character.

Note: Application remarks must be unique across the domain.

Application Information Level 2

The level 2 structure allows you to get or set information about an application's fixed data only.

```
struct app_info_2 {
    unsigned char                app2_name[APP_LEN + 1];
    unsigned char                app2_pad_1;
    unsigned char LSFAR * LSPTR  app2_remark;
    unsigned char LSFAR * LSPTR  app2_command;
    unsigned char LSFAR * LSPTR  app2_command_parms;
    unsigned char                app2_app_alias_or_drv[ALIAS_LEN + 1];
    unsigned char                app2_pad_2;
    unsigned char                app2_app_drive;
    unsigned char LSFAR * LSPTR  app2_app_path_to_dir;
    unsigned char                app2_wrkdir_alias_or_drv[ALIAS_LEN + 1];
    unsigned char                app2_pad_3;
    unsigned char                app2_wrkdir_drive;
    unsigned char LSFAR * LSPTR  app2_wrkdir_path_to_dir;
    unsigned char                app2_prompt;
    unsigned char                app2_interface;
    unsigned char                app2_apptype;
};
```

where:

- *app2_name* is an ASCIIZ string that specifies an identifying name for an application. DCDB.H defines bytes that are not valid in an application name. See [Application Information Level 0](#).
- *app2_pad_1* word-aligns the data structure component.
- *app2_remark* points to an ASCIIZ string that provides a description of the application. The string must not be longer than 40 bytes. The string appears as the application's title in the applications folder. A null string is not allowed; a remark is required. The remark string cannot include a backslash (\) character.

Note: Application remarks must be unique across the domain.

- *app2_command* points to an ASCIIZ string that indicates the command that starts the application. A null string is not allowed; a command is required. For DOS applications, the length of *app2_command* plus *app2_command_parms* plus *app2_app_path_to_dir* must not exceed 89 bytes. For OS/2 applications, the length of *app2_command* plus *app2_command_parms* must not exceed 119 bytes.
- *app2_command_parms* points to an ASCIIZ string that indicates any fixed parameters needed for this application. The parameters must be delimited by spaces. A null string indicates that there are no fixed parameters for the application.
- *app2_app_alias_or_drv* specifies the alias or drive where the application resides. If the application resides on the user's local workstation, specify a drive letter, followed by a colon (:). If the application resides on a server, specify an existing alias.
- *app2_pad_2* word-aligns the data structure component.
- *app2_app_drive* is the drive letter used when connecting to a remote resource where the application resides. This field is required for any private DLS application only if an alias is specified instead of a drive letter in the former *app2_app_alias_or_drv* field. This is optional for an OS/2 application when an alias is specified in the *app2_app_alias_or_drv* field, and ignored when that field contains a drive letter.

- *app2_app_path_to_dir* is required and points to an ASCIIZ string containing the remaining path to the application, relative to the information in *app2_app_alias_or_drv*. The string must begin with a backslash (\).
- *app2_wrkdir_alias_or_drv* specifies the directory that is made current when the application runs. If the working directory is on the local workstation, this field specifies the drive letter, followed by a colon (:) where the directory is located. If the working directory is remote, this field specifies an existing alias where the directory is located. A null string indicates that no working directory is specified.
- *app2_pad_3* word-aligns the data structure component.
- *app2_wrkdir_drive* specifies the drive to which the working directory is to be assigned when the application is started. For OS/2 applications, this can be any letter D-Z. For DOS applications, this can be any letter A-X. A value of an asterisk (*) indicates that the system should choose a drive when the application is started. If the working directory is local, this field must be set to 0.
- *app2_wrkdir_path_to_dir* points to an ASCIIZ string that specifies the remaining path to the working directory, relative to the information in *app2_wrkdir_alias_or_drv*. The string must begin with a backslash (\). If *app2_wrkdir_alias_or_drv* is NULL, this field must be set to 0.
- *app2_prompt* indicates whether the user is to be prompted for parameters when the application starts. A value of 1 indicates that the user is to be prompted for parameters; a value of 0 indicates that the user is not to be prompted.
- *app2_interface* is used for OS/2 applications only and is ignored for DOS. Defined in DCDB.H, it can be one of three values:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_PM	0x1	The application is a windowed program running the Presentation Manager ace.
APP_PROT	0x2	The application is a protected mode program that is not windowed.
APP_VIO	0x4	The application is a windowed program that is not PM and that uses OS/2 video I/O functions.

- *app2_apptype* is one of four values defined as follows in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_DOS_PUBLIC	0x1	Public DOS applications
APP_OS2_PUBLIC	0x2	Public OS/2 applications
APP_OS2_PRIVATE	0x4	Private OS/2 applications
APP_DOS_PRIVATE	0x8	Private DOS applications

Application Information Level 3

The level 3 structure allows you to enumerate, get, or set information about an application's fixed data and its list data.

```
struct app_info_3{
    unsigned char          app3_name[APP_LEN + 1];
    unsigned char          app3_pad_1;
    unsigned char LSFAR * LSPTR app3_remark;
    unsigned char LSFAR * LSPTR app3_command;
    unsigned char LSFAR * LSPTR app3_command_parms;
    unsigned char          app3_app_alias_or_drv[ALIAS_LEN + 1];
    unsigned char          app3_pad_2;
```

```

unsigned char      app3_app_drive;
unsigned char LSFAR * LSPTR app3_app_path_to_dir;
unsigned char      app3_wrkdir_alias_or_drv[ALIAS_LEN + 1];
unsigned char      app3_pad_3;
unsigned char      app3_wrkdir_drive;
unsigned char LSFAR * LSPTR app3_wrkdir_path_to_dir;
unsigned char      app3_prompt;
unsigned char      app3_interface;
unsigned char      app3_apptype;
unsigned short     app3_res_count;
};

```

where:

- The first 11 fields in this data structure are identical to those in the previous level.
- *app3_res_count* is the number of *app_res_list* structures that immediately follow the *app_info_3* structure. A value of 0 indicates that the application does not require any redirected devices when it runs.

Application Information Level 10

Application level 10 is structured as follows.

```

struct app_info_10 {
    unsigned char  LSFAR * LSPTR  app10_name;
    unsigned char  LSFAR * LSPTR  app10_globalname;
    unsigned char  app10_pad;
};

```

where:

- *app10_name* is an ASCIIZ string that specifies an identifying name for the application. You can use any characters except embedded blanks and the following:
" / \ [] : | < > + = ; , . ? *
- *app10_globalname* is universally meaningful and usable from anywhere in the DCE naming environment. The prefix /... or /.: indicates that a name is global.
- *app10_pad* word-aligns the data structure component.

Application Information Level 11

```

struct app_info_11 {
    unsigned char  LSFAR * LSPTR  app11_name;
    unsigned char  LSFAR * LSPTR  app11_globalname;
    unsigned char  app11_pad[2];
    unsigned char  LSFAR * LSPTR  app11_remark;
};

```

where:

- The first field in this data structure is identical to that in the previous level.
- *app11_pad[2]* word-aligns the data structure component.

- *app11_remark* points to an ASCIIZ string that provides a description of the application. The string must not be longer than 40 bytes. The string displays as the application title in the applications folder. Null strings are not allowed; a remark is required. The remark string cannot include a backslash (\) character.

Note: Application remarks must be unique across the domain.

Application Information Level 12

The level 12 structure allows you to get or set information about an application's fixed data only.

```
struct app_info_12 {
    unsigned char    LSFAR * LSPTR    app12_name;
    unsigned char    LSFAR * LSPTR    app12_globalname;
    unsigned char    LSFAR * LSPTR    app12_pad_1[2];
    unsigned char    LSFAR * LSPTR    app12_remark;
    unsigned char    LSFAR * LSPTR    app12_command;
    unsigned char    LSFAR * LSPTR    app12_command_parms;
    unsigned char    LSFAR * LSPTR    app12_app_alias_or_drv;
    unsigned char    LSFAR * LSPTR    app12_pad_2[6];
    unsigned char    LSFAR * LSPTR    app12_app_drive;
    unsigned char    LSFAR * LSPTR    app12_app_path_to_dir;
    unsigned char    LSFAR * LSPTR    app12_wrkdir_alias_or_drv;
    unsigned char    LSFAR * LSPTR    app12_pad_3[6];
    unsigned char    LSFAR * LSPTR    app12_wrkdir_drive;
    unsigned char    LSFAR * LSPTR    app12_wrkdir_path_to_dir;
    unsigned char    LSFAR * LSPTR    app12_prompt;
    unsigned char    LSFAR * LSPTR    app12_interface;
    unsigned char    LSFAR * LSPTR    app12_apptype;
};
```

where:

- *app12_name* is an ASCIIZ string that specifies an identifying name for an application. DCDB.H defines bytes that are not valid in an application name. See [Application Information Level 0](#).
- *app12_pad_1* word-aligns the data structure component.
- *app12_remark* points to an ASCIIZ string that provides a description of the application. The string must not be longer than 40 bytes. The string displays as the application title in the applications folder. A null string is not allowed; a remark is required. The remark string cannot include a backslash (\) character.

Note: Application remarks must be unique across the domain for OS/2 LAN Server, OS/2 Warp Server and across the resource domain for DSS

- *app12_command* points to an ASCIIZ string that indicates the command that starts the application. A null string is not allowed; a command is required. For DOS applications, the length of *app12_command* plus *app12_command_parms* plus *app12_app_path_to_dir* must not exceed 89 bytes. For OS/2 applications, the length of *app12_command* plus *app12_command_parms* must not exceed 119 bytes.
- *app12_command_parms* points to an ASCIIZ string that indicates any fixed parameters needed for this application. The parameters must be delimited by spaces. A null string indicates that there are no fixed parameters for the application.
- *app12_app_alias_or_drv* specifies the alias or drive where the application resides. If the application resides on the user's local workstation, specify a drive letter, followed by a colon (:). If the application resides on a server, specify an existing alias. Alias name may be just the alias name by itself if the alias is defined in the resource domain of the application definition. Or you may use the global name or extended alias name. The global name needs the cell name, resource domain name and the alias name along with the necessary directory names. The extended name uses only the alias with @ then the resource name. If the cell name is different than the local cell, then use another @ and the cell name.
- *app12_pad_2* word-aligns the data structure component.
- *app12_app_drive* is the drive letter used when connecting to a remote resource where the application resides. This field is required for any private DLS application only if an alias is specified instead of a drive letter in the former *app12_app_alias_or_drv* field. This is optional for an OS/2 application when an alias is specified in the *app12_app_alias_or_drv* field, and ignored when that field contains a drive letter.

- *app12_app_path_to_dir* is required and points to an ASCIIZ string containing the remaining path to the application, relative to the information in *app12_app_alias_or_drv*. The string must begin with a backslash (\).
- *app12_wrkdir_alias_or_drv* specifies the directory that is made current when the application runs. If the working directory is on the local workstation, this field specifies the drive letter, followed by a colon (:) where the directory is located. If the working directory is remote, this field specifies an existing alias where the directory is located. A null string indicates that no working directory is specified. Alias name may be just the alias name by itself if the alias is defined in the resource domain of the application definition. Or you may use the global name or extended alias name. The global name needs the cell name, resource domain name and the alias name along with the necessary directory names. The extended name uses only the alias with @ then the resource name. If the cell name is different than the local cell, then use another @ and the cell name.
- *app12_pad_3* word-aligns the data structure component.
- *app12_wrkdir_drive* specifies the drive to which the working directory is to be assigned when the application is started. For OS/2 applications, this can be any letter D-Z. For DOS applications, this can be any letter A-X. A value of an asterisk (*) indicates that the system should choose a drive when the application is started. If the working directory is local, this field must be set to 0.
- *app12_wrkdir_path_to_dir* points to an ASCIIZ string that specifies the remaining path to the working directory, relative to the information in *app_x_wrkdir_alias_or_drv*. The string must begin with a backslash (\). If *app12_wrkdir_alias_or_drv* is NULL, this field must be set to 0.
- *app12_prompt* indicates whether the user is to be prompted for parameters when the application starts. A value of 1 indicates that the user is to be prompted for parameters; a value of 0 indicates that the user is not to be prompted.
- *app12_interface* is used for OS/2 applications only and is ignored for DOS. Defined in DCDB.H header file, it can be one of three values:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_PM	0x1	The application is a windowed program running the Presentation Manager interface.
APP_PROT	0x2	The application is a protected mode program that is not windowed.
APP_VIO	0x4	The application is a windowed program that is not PM and that uses OS/2 video I/O functions.

- *app12_apptype* is one of four values defined as follows in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_DOS_PUBLIC	0x1	Public DOS applications
APP_OS2_PUBLIC	0x2	Public OS/2 applications
APP_OS2_PRIVATE	0x4	Private OS/2 applications
APP_DOS_PRIVATE	0x8	Private DOS applications

Application Information Level 13

The level 13 structure allows you to enumerate, get, or set information about the fixed data and list data in an application.

```
struct app_info_13{
    unsigned char    LSFAR * LSPTR    app13_name;
```

```

unsigned char LSFAR * LSPTR app13_globalname;
unsigned char app13_pad_1[2];
unsigned char LSFAR * LSPTR app13_remark;
unsigned char LSFAR * LSPTR app13_command;
unsigned char LSFAR * LSPTR app13_command_parms;
unsigned char LSFAR * LSPTR app13_app_alias_or_drv;
unsigned char app13_pad_2[2];
unsigned char app13_app_drive;
unsigned char LSFAR * LSPTR app13_app_path_to_dir;
unsigned char LSFAR * LSPTR app13_wrkdir_alias_or_drv;
unsigned char app13_pad_3[6];
unsigned char app13_wrkdir_drive;
unsigned char LSFAR * LSPTR app13_wrkdir_path_to_dir;
unsigned char app13_prompt;
unsigned char app13_interface;
unsigned char app13_apptype;
unsigned short app13_res_count;
};

```

where:

- The first 11 fields in this data structure are identical to those in the previous level.
- *app13_res_count* is the number of *app_res_list* structures that immediately follow the *app_info_3* structure. A value of 0 indicates that the application does not require any redirected devices when it runs.

Application Resource Data Structure

Directory and Security Server Only

Used only with *app_info_13*.

```

struct app_res_list_10 {
    unsigned char LSFAR * LSPTR arl10_alias;
    unsigned char arl10_pad_1[6 ];
    unsigned char arl10_device[DEVLEN + 1];
};

```

LAN Server and OS/2 Warp Server Only

Used only with *app_info_3*.

```

struct app_res_list {
    unsigned char arl_alias[ALIAS_LEN + 1];
    unsigned char arl_pad_1;
    unsigned char arl_device[DEVLEN + 1];
};

```

where:

- *arl_alias* specifies the alias for the resource required by the application. The alias must exist. Alias name may be just the alias name by itself if the alias is defined in the resource domain of the application definition. Or you may use the global name or extended alias name. The global name needs the cell name, resource domain name and the alias name along with the necessary directory names. The extended name uses only the alias with @ and the resource name. If the cell name is different than the local cell, then use another @ and the cell name.
- *arl_pad_1* word-aligns the data structure component.
- *arl_device* indicates the device assigned to the resource specified by *arl_alias* when the application starts. If *arl_alias* specifies a file alias, *arl_device* must be a drive letter followed by a colon (:). Valid drive letters for OS/2 applications are D-Z. Valid drive letters for DOS applications are A-X. If *arl_alias* specifies a print alias, valid values for *arl_device* are LPT1-LPT9. If *arl_alias* specifies a serial device alias, valid values for *arl_device* are LPT1-LPT9 and COM1-COM16.

Note: Additionally, each *app_res_list* structure must have a unique value for its *arl_device* field. For example, if *arl_alias* specifies a file alias, the drive letter specified in the *arl_device* field must be different from *app_x_app_drive*, *app_x_wrkdir_drive*, and any other *arl_device* fields for file aliases.

NetAppAdd or Net32AppAdd

NetAppAdd or Net32AppAdd

The NetAppAdd API adds an application definition.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAppAdd requires one of the following:

- The calling process has Administrator authority.
- The calling process has server operator privilege.
- The calling process has User authority and is adding a private application for the logged on user.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAppAdd(pszTarget, pszUserID, usLevel,
          buf, usBuflen);  /* 16 bit */

Net32AppAdd(pszTarget, pszUserID, ulLevel,
            buf, ulBuflen, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.	
<i>null or lservername</i>		added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomainname</i>		added to resource domain of the local cell when connected to a DSS cell.
<i>//resdomainname@cellname</i>		added to the resource domain of the specified

cellname when
connected to a
DSS cell.

sLevel or *ulLevel*

(16-bit short or 32-bit unsigned long) must be 3, which specifies the data structure described in [Application Information Level 3](#). For DSS see [Application Information Level 3](#) or [Application Information Level 13](#).

pStatusbuf

must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS may point to a buffer of size `lsdce_err_stat_t` (defined in `lsdceerr.h`). Contains specific DCE error information when `NERR_DCEError` is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
<code>ERROR_FILE_NOT_FOUND</code>	2	The file was not found.
<code>ERROR_TOO_MANY_OPEN_FILES</code>	4	The maximum number of open files was exceeded.
<code>ERROR_ACCESS_DENIED</code>	5	Administrator privilege is required.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	8	Sufficient memory is not available.
<code>ERROR_INVALID_DATA</code>	13	The specified data is not valid.
<code>ERROR_INVALID_DRIVE</code>	15	The specified drive is not valid.
<code>ERROR_BAD_LENGTH</code>	24	The length is not valid.
<code>ERROR_ALREADY_ASSIGNED</code>	85	Duplicate redirection.
<code>ERROR_INVALID_LEVEL</code>	124	The <code>sLevel</code> parameter is not valid.
<code>ERROR_BAD_PATHNAME</code>	161	Path name not valid.
<code>NERR_NetNotStarted</code>	2102	The redirector <code>NETWKSTA.200</code> has not been started.
<code>NERR_BufTooSmall</code>	2123	The buffer is too small for fixed-length data.
<code>NERR_InvalidAPI</code>	2142	The requested API is not supported on the remote server.
<code>NERR_UserNotFound</code>	2221	The user name cannot be found.
<code>NERR_AliasNotFound</code>	2783	An alias specified in the input buffer cannot be found.
<code>NERR_DCEError</code>	2786	A domain control database file is unreadable or cannot be accessed at this time.
<code>NERR_AppExists</code>	2792	The application to be added already exists.
<code>NERR_NotPrimaryDCDB</code>	2795	An attempt was made to access a domain control database file on a machine that

		is not the domain controller.
NERR_BadAppRemark	2796	The application remark contains an error or is not unique.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_InvalidAliasNameForma	7531	The syntax of the alias name is not valid.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddress

Remarks

LAN Server and Warp Server Only

If adding a private application, the domain control database subdirectories and files must exist for the user. The NetUserDCDBInit API can be used to initialize these subdirectories and files.

Related Information

For information about:

- Initializing a particular user's domain control database files, see [User - NetUserDCDBInit](#).
- Listing application definitions, see [Application - NetAppEnum](#).
- Changing an application definition, see [Application - NetAppSetInfo](#).
- Removing an application definition, see [Application - NetAppDel](#).
- Listing applications that appear in the folders of a particular user's desktop, see [User - NetUserGetAppSel](#).
- Changing the list of applications that appear in the folders of a particular user's desktop, see [User - NetUserSetAppSel](#).

NetAppDel or Net32AppDel

NetAppDel or Net32AppDel

The NetAppDel API deletes an application definition.

The NetAppSetInfo API can be used to delete an application's list data (redirections associated with an application).

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAppDel requires one of the following:

- The calling process has Administrator authority .
- The calling process has server operator privilege.
- The calling process has User authority and is deleting a private application for the logged-on user.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAppDel(pszTarget, pszUserID, pszAppname,
          reserved);      /* 16 bit */

Net32AppDel(pszTarget, pszUserID, pszAppname,
            reserved, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.
<i>null or lservername</i>	added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomname</i>	added to resource domain of the local cell when connected to a DSS cell.
<i>//resdomname@cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>pszAppname</i>	(unsigned char LSFAR *) points to an ASCIIZ string indicating the application to delete. For DSS the application name may be a global name.
<i>reserved</i>	(unsigned long) is reserved and must be set to 0.
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For

DSS may point to a buffer of size `lsdce_err_stat_t` (defined in `lsdceerr.h`). Contains specific DCE error information when `NERR_DCEError` is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
<code>ERROR_FILE_NOT_FOUND</code>	2	The file was not found.
<code>ERROR_TOO_MANY_OPEN_FILES</code>	4	The maximum number of open files was exceeded.
<code>ERROR_ACCESS_DENIED</code>	5	Administrator privilege is required.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	8	Sufficient memory is not available.
<code>ERROR_INVALID_DATA</code>	13	The specified data is not valid.
<code>ERROR_INVALID_PARAMETER</code>	87	At least one parameter value is not valid.
<code>NERR_NetNotStarted</code>	2102	The redirector <code>NETWKSTA.200</code> has not been started.
<code>NERR_BufTooSmall</code>	2123	The buffer is too small for fixed-length data.
<code>NERR_InvalidAPI</code>	2142	The requested API is not supported on the remote server.
<code>NERR_UserNotFound</code>	2221	The user name cannot be found.
<code>NERR_DCEError</code>	2786	A domain control database file is unreadable or cannot be accessed at this time.
<code>NERR_AppNotFound</code>	2793	The application to be deleted was not found.
<code>NERR_NotPrimaryDCDB</code>	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
<code>NERR_DCEError</code>	7500	DCE error status returned.
<code>NERR_DCEErrorLogged</code>	7501	DCE error status has been logged.
<code>NERR_ResDomNotFound</code>	7503	The specified resource domain does not exist.
<code>NERR_InvalidObjGlobalName</code>	7504	The specified object global name is not valid.
<code>NERR_InvalidResDomName</code>	7505	The specified resource domain name is not valid.
<code>NERR_AuthTicketExpired</code>	7511	The user's DCE authorization ticket has expired.

NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
-----------------------	------	--

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Related Information

For information about:

- Creating an application definition, see [Application - NetAppAdd](#).
- Listing application definitions, see [Application - NetAppEnum](#).
- Changing an application definition, see [Application - NetAppSetInfo](#).
- Listing applications that appear in folders on a particular user's desktop, see [User - NetUserGetAppSel](#).
- Changing the list of applications that appear in a particular user's desktop folder, see [User - NetUserSetAppSel](#).

NetAppEnum or Net32AppEnum

NetAppEnum or Net32AppEnum

The NetAppEnum API returns information about applications according to the assigned user ID and whether the application is public or private.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call may also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAppEnum requires one of the following:

- The calling process has Administrator authority .
- The calling process has server operator privilege.
- The calling process has User authority and is deleting a private application for the logged-on user.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAppEnum(pszTarget, pszUserID, usLevel, type, buf,
           usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32AppEnum(pszTarget, pszUserID, ulLevel, type, buf,
             ulBuflen, pulEntriesReturned, pulEntriesAvail,
             pStatusbuf);                                     /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszTarget

is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.

null or lservername

added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.

//resdomname

added to resource domain of the local cell when connected to a DSS cell.

//resdomname@cellname

added to the resource domain of the specified cellname when connected to a DSS cell.

sLevel or ulLevel

(16-bit short or 32-bit unsigned long) can be 0, 1, 2, or 3, specifying which data structure to use, as described in [Application Information Level 0](#) through [Application Information Level 3](#).

For DSS see [Application Information Level 10](#), [Application Information Level 11](#), [Application Information Level 12](#) or [Application Information Level 13](#).

type

(16-bit unsigned short or 32-bit unsigned long) specifies the kind of application to be enumerated, as defined in the DCDB.H header file:

SYMBOLIC CONSTANT	BIT MASK	MEANING
APP_DOS_PUBLIC	0x1	Public DOS app.
APP_OS2_PUBLIC	0x2	Public OS/2 app.
APP_OS2_PRIVATE	0x4	Private OS/2 app.
APP_DOS_PRIVATE	0x8	Private DOS app.

pStatusbuf

must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size `lsc_err_stat_t` (defined in `lscerr.h`). Contains specific DCE error information when `NERR_DCEError` is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.

ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_AppNotFound	2793	The application to be deleted was not found.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotfound	7503	The specified resource domain does not exist.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

If both public and private applications are to be enumerated, the return buffer contains public applications followed by private applications.

If both public and private applications are to be enumerated, the *pszUser/D* parameter must point to a string indicating

the user ID of the user whose private applications are to be enumerated.

LAN Server and Warp Server Only

If enumerating private applications, the domain control database subdirectories and files must exist for the user. The NetUserDCDBInit API can be used to initialize these subdirectories and files.

Related Information

For information about:

- Retrieving information for a particular application definition, see [Application - NetAppGetInfo](#).
- Listing applications that appear in a particular user's desktop application folders, see [User - NetUserGetAppSel](#).
- Initializing a particular user's domain control database files, see [User - NetUserDCDBInit](#).

NetAppGetInfo or Net32AppGetInfo

NetAppGetInfo or Net32AppGetInfo

The NetAppGetInfo API retrieves information about an application.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote can also be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetGetInfo requires one of the following:

- The calling process has Administrator authority.
- The calling process has server operator privilege.
- The calling process has User authority and is deleting a private application for the logged-on user.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAppGetInfo(pszTarget, pszUserID, pszAppname, usLevel,
             buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32AppGetInfo(pszServername, pszUserID, pszAppname, ulLevel,
               buf, ulBuflen, pulBytesAvail, pStatusbuf);
                                                    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszTarget

is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.

null or lservername

added to DCDB of a specified server when connected to a

		LAN Server or OS/2 LAN Server or OS/2 Warp Server.
	<i>//resdomname</i>	added to resource domain of the local cell when connected to a DSS cell.
	<i>//resdomname@cellname</i>	added to the resource domain of the specified cellname when connected to a DSS cell.
<i>pszAppname</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the application for which information is to be retrieved. For DSS, the application name may be a global name.	
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0, 1, 2, or 3, specifying which data structure to use, as described in Application Information Level 0 through Application Information Level 3 . For DSS see Application Information Level 10 , Application Information Level 11 , Application Information Level 12 or Application Information Level 13 .	
<i>pStatusbuf</i>	must be NULL for OS/2 LAN Server and OS/2 Warp Server requests. For DSS, may point to a buffer of size <code>lsdce_err_stat_t</code> (defined in <code>lsdceerr.h</code>). Contains specific DCE error information when <code>NERR_DCEError</code> is returned.	

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
<code>ERROR_FILE_NOT_FOUND</code>	2	The file was not found.
<code>ERROR_TOO_MANY_OPEN_FILES</code>	4	The maximum number of open files was exceeded.
<code>ERROR_ACCESS_DENIED</code>	5	Administrator privilege is required.
<code>ERROR_NOT_ENOUGH_MEMORY</code>	8	Sufficient memory is not available.
<code>ERROR_INVALID_DATA</code>	13	The specified data is not valid.
<code>ERROR_INVALID_LEVEL</code>	124	The <code>sLevel</code> parameter is not valid.
<code>ERROR_MORE_DATA</code>	234	Additional data is available, but the buffer is too small.
<code>NERR_NetNotStarted</code>	2102	The redirector <code>NETWKSTA.200</code> has not been started.
<code>NERR_BufTooSmall</code>	2123	The buffer is too small for fixed-length data.
<code>NERR_InvalidAPI</code>	2142	The requested API is not supported on

the remote server.

NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotFound	2793	The application was not found.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_IncompatibleLevel	7532	The level specified is incompatible for this resource..

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

LAN Server and Warp Server Only

When getting information about private applications, the domain control database subdirectories and files must exist for the user. The NetUserDCDBInit API can be used to initialize these subdirectories and files.

Related Information

For information about:

- Initializing a particular user's domain control database files, see [User - NetUserDCDBInit](#).
- Changing an application definition, see [Application - NetAppSetInfo](#).
- Listing application definitions, see [Application - NetAppEnum](#).
- Listing applications that appear in a particular user's desktop application folders, see [User - NetUserGetAppSel](#).

NetAppSetInfo or Net32AppSetInfo

NetAppSetInfo or Net32AppSetInfo

The NetAppSetInfo API changes information for an application.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This remote call can so be issued to an OS/2 Warp Server or Directory and Security Server workstation.

A successful call to NetAppSetInfo requires one of the following:

- The calling process has Administrator authority.
- The calling process has server operator privilege.
- The calling process has User authority and is setting information about the logged-on user's private applications.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>      /* DSS only */

NetAppSetInfo(pszTarget, pszUserID, pszAppname, usLevel,
              buf, usBuflen, parmnum);      /* 16 bit */

Net32AppSetInfo(pszTarget, pszUserID, pszAppname, ulLevel,
                buf, ulBuflen, parmnum, pStatusbuf);
                                                    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.	
<i>null or llservname</i>		added to DCDB of a specified server when connected to a LAN Server or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomname</i>		added to resource domain of the local cell when connected to a DSS cell.
<i>//resdomname@cellname</i>		added to the resource domain of the specified cellname when

		started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_AliasNotFound	2783	An alias specified in the input buffer cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotFound	2793	The application was not found.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_BadAppRemark	2796	The application remark contains an error or is not unique.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_ResDomNotfound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_InvalidAliasNameForma	7531	The syntax of the alias name is not valid.
NERR_IncompatibleLevel	7532	The level specified is incompatible for this resource..

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

To delete an application's list data, call NetAppSetInfo at level 3 with *app3_app_res_count* set to 0.

LAN Server and Warp Server Only

When setting information about private applications, the domain control database subdirectories and files must exist for

the user. The NetUserDCDBInit API can be used to initialize these subdirectories and files.

Related Information

For information about:

- Initializing a particular user's domain control database files, see [User - NetUserDCDBInit](#).
- Changing the list of applications that appear in a particular user's desktop application folders, see [User - NetUserSetAppSel](#).
- Retrieving information about a particular application definition, see [Application - NetAppGetInfo](#).

Auditing Category

This category includes the following APIs:

[Auditing - NetAuditClear](#)
[Auditing - NetAuditRead](#)
[Auditing - NetAuditWrite](#)

Auditing APIs control the *audit log file*, which contains an audit trail of operations that occur on a server. These functions are used with the AUDIT.H and NETCONS.H header files.

Each time a user or application connects to or disconnects from resources on a server, an audit entry can be generated to record the connection or disconnection. Audit entries are stored in an ASCII file. The default audit log file is \BMLAN\LOGS\NET.AUD. All of the auditing functions perform their operations on this file.

Note: The auditing functions only control changing the contents of the audit log file. To read the audit log file, an application first must call the NetAuditRead API to get the handle of the file. The DosRead API then can be called to read the file. To close the file, an application must call the DosClose API.

LAN Server provides for the following types of audit entries:

- A change in the status of a server
- The beginning of a session
- The end of a session
- A password error
- The start of a connection
- A disconnection
- A rejected connection request
- An access made to a resource
- The rejection of an access
- The closing of a file, device, or pipe
- A change to a service status code or text
- A change to an access control profile
- A change to the UAS database
- The logon of a user
- The logoff of a user
- The denial of a logon
- The limit of an account exceeded

Applications can create additional types of audit entries with the NetAuditWrite API. The other two auditing APIs open (NetAuditRead) and clear (NetAuditClear) the audit log file.

Directory and Security Server Only

The DSS Security and Time servers use audit services provided by DCE. To access the audit trails, you must use the DCE dce_and RPCs rather than the NetAudit APIs.

Also see the *IBM Directory and Security Server Administrator's Guide* for more information about changes in auditing on DSS.

DOS Considerations

Under DOS, NetAuditClear and NetAuditWrite can be run only on a remote LAN Server workstation.

Audit Entry Data Structures

Audit entry data structures consist of a fixed-length header followed by 0 or more variable-length audit entry data structures.

In addition to the information contained within it, a data structure can specify that an additional variable-length ASCIIZ string is to be included. To do this, an unsigned short offset is used, where the address of the additional ASCIIZ string is equal to the address of the variable-length structure plus the offset.

For example, assume that *ap* points to a buffer containing a complete audit entry and that *ae_type* contains the value AE_CONNSTOP, specifying the predefined *ae_connstop* data structure. To make the variable *computer_name* point to the ASCIIZ string containing the name of the client whose connection was stopped, an application would perform the following algorithm:

```
struct audit_entry * ap;           /* fixed portion of audit entry */
struct ae_connstop * acp;         /* variable-length structure */
unsigned char * computer_name;     /* pointer to variable-length string */
                                /* calculate offset to variable-length struct */

acp = (struct ae_connstop *) ((unsigned char *) ap + ap->ae_data_offset);

                                /* calculate offset to computer name */

computer_name = (unsigned char *) acp + acp->ae_cp_compname;
```

Some of the structures point to more than one data item. To ensure that an item is used, the programmer must test for a null pointer and for a null string. For example, many audit data structures have a user name and a computer name. If the user name is not significant, then the computer name must be used instead. To determine if the user name is significant, check whether *ae_so_username* is 0 (null pointer) or whether it points to a 0 (null string).

Audit Entry Header

Audit entry data structures begin with a fixed-length header, defined as follows:

```
struct audit_entry {
    unsigned short ae_len;
    unsigned short ae_reserved;
    unsigned long  ae_time;
    unsigned short ae_type;
    unsigned short ae_data_offset; /* offset from beginning address
                                /* of audit_entry to ae_data */
};
```

Variable-length data specific to the type of audit entry follows:

```
    unsigned char  ae_data[];
    unsigned short ae_len2;
```

where:

- *ae_len* and *ae_len2* specify the length of the audit entry. (The *ae_len* is included at both the beginning and the end of the audit entry to enable both backward and forward scanning of the file.) To calculate the entry size, add the size of the *audit_entry* data structure to the size of the variable-length *ae_data* and the size of an unsigned short integer, as follows:

```
totalsize = sizeof (struct audit_entry) +
```

sizeof (ae_data) + sizeof (unsigned short)

- *ae_reserved* must be 0.
- *ae_time* is a time stamp indicating the time the audit file log entry was made.
- *ae_type* indicates the type of audit entry. Type values ranging from 0x0000 through 0x07FF are reserved. The AUDIT.H header file defines the following types of data entries:

SYMBOLIC CONSTANT	VALUE	PURPOSE
AE_SRVSTATUS	0	Status of server changed.
AE_SESSLOGON	1	Session logged on.
AE_SESSLOGOFF	2	Session logged off.
AE_SESSPWERR	3	Password error.
AE_CONNSTART	4	Connection started.
AE_CONNSTOP	5	Connection stopped.
AE_CONNREJ	6	Connection rejected.
AE_RESACCESS	7	Access granted.
AE_RESACCESSREJ	8	Access rejected.
AE_CLOSEFILE	9	File, device, or pipe closed.
AE_SERVICESTAT	11	Service status code or text changed.
AE_ACLMOD	12	Access control profile changed.
AE_UASMOD	13	User accounts subsystem database changed.
AE_NETLOGON	14	User logged on to the network.
AE_NETLOGOFF	15	User logged off of the network.
AE_NETLOGDENIED	16	Network logon denied.
AE_ACCLIMITEXCD	17	Account limit exceeded.
AE_RESACCESS2	18	Access granted.
AE_ACLMODFAIL	19	Access control list (ACL) change failed.

- *ae_data_offset* specifies the byte offset from the beginning of the audit entry to the start of the variable-length portion (*ae_data*) of the audit entry. To calculate the start of *ae_data*, add the value of *ae_data_offset* to the starting address of the fixed-length portion of the entry.
- *ae_data* is the variable-length portion of the audit entry, which differs depending on the type of entry specified by *ae_type*. The information starts at *ae_data_offset* bytes from the top of the audit entry. See the following section for information about the structure of each entry type that the LAN Server software defines.

Audit Server Status

```
struct ae_srvstatus {
    unsigned short ae_sv_status;
};
```

where *ae_sv_status* is one of four values indicating a status of the server. These values, defined in the AUDIT.H header file, mean the following:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_SRVSTART	0	Server software started.
AE_SRVPAUSED	1	Server software paused.
AE_SRVCONT	2	Server software restarted.
AE_SRVSTOP	3	Server software stopped.

Audit Logon

```
struct ae_sesslogon {
    unsigned short ae_so_compname; /* offset */
    unsigned short ae_so_username; /* offset */
    unsigned short ae_so_privilege;
};
```

where:

- ae_so_compname* is an offset (from the beginning address of the *ae_sesslogon* data structure) to an ASCIIZ string, indicating the requester that established the session.
- ae_so_username* is an offset (from the beginning address of the *ae_sesslogon* data structure) to an ASCIIZ string, indicating the name of the user who initiated the session. If 0, *ae_so_username* and *ae_so_compname* are the same.
- ae_so_privilege* is one of three values specifying the permission level assigned to *ae_so_username*. These values, defined in the AUDIT.H header file, have the following meanings:

SYMBOLIC CONSTANT	VALUE	PRIVILEGE
AE_GUEST	0	Guest
AE_USER	1	User
AE_ADMIN	2	Administrator

Audit Logoff


```
struct ae_sesslogoff {
    unsigned short ae_sf_compname; /* offset */
    unsigned short ae_sf_username; /* offset */
    unsigned short ae_sf_reason;
};
```

where:

- *ae_sf_compname* is an offset (from the beginning address of the *ae_sesslogoff* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_sf_username* is an offset (from the beginning address of the *ae_sesslogoff* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_sf_username* and *ae_sf_compname* are the same.
- *ae_sf_reason* is one of five values indicating why the session was disconnected. These values, defined in the AUDIT.H header file, mean the following:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_NORMAL	0	Normal disconnection or user name limit.
AE_ERROR	1	Error, session disconnect, or bad password.
AE_AUTODIS	2	Autodisconnect (timeout), share removed, or administrative permissions required.
AE_ADMINDIS	3	Administrative disconnection (forced).
AE_ACCRESTRICT	4	Forced off by account system because of account restriction, such as logon hours.

Audit Password

```
struct ae_sesspwerr {
    unsigned short ae_sp_compname; /* offset */
    unsigned short ae_sp_username; /* offset */
};
```

where:

- *ae_sp_compname* is an offset (from the beginning address of the *ae_sesspwerr* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_sp_username* is an offset (from the beginning address of the *ae_sesspwerr* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_sp_username* and *ae_sp_compname* are the same.

Audit Connection Start

```
struct ae_connstart {
```

```

unsigned short ae_ct_compname; /* offset */
unsigned short ae_ct_username; /* offset */
unsigned short ae_ct_netname; /* offset */
unsigned short ae_ct_connid;
};

```

where:

- *ae_ct_compname* is an offset (from the beginning address of the *ae_connstart* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_ct_username* is an offset (from the beginning address of the *ae_connstart* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_ct_username* and *ae_ct_compname* are the same.
- *ae_ct_netname* is an offset (from the beginning address of the *ae_connstart* data structure) to an ASCIIZ string indicating the netname of the resource with which the connection was made.
- *ae_ct_connid* is the connection identification number.

Audit Connection Stop

```

struct ae_connstop {
    unsigned short ae_cp_compname; /* offset */
    unsigned short ae_cp_username; /* offset */
    unsigned short ae_cp_netname; /* offset */
    unsigned short ae_cp_connid;
    unsigned short ae_cp_reason;
};

```

where:

- *ae_cp_compname* is an offset (from the beginning address of the *ae_connstop* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_cp_username* is an offset (from the beginning address of the *ae_connstop* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_cp_username* and *ae_cp_compname* are the same.
- *ae_cp_netname* is an offset (from the beginning address of the *ae_connstop* data structure) to an ASCIIZ string indicating the connected netname of the resource.
- *ae_cp_connid* is the connection identification number.
- *ae_cp_reason* is one of three values indicating why the session was disconnected. These values, defined in the AUDIT.H header file, mean the following:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_NORMAL	0	Normal disconnection, or user name limit.
AE_SESSDIS	1	Error, session disconnect, or bad password.
AE_UNSHARE	2	Autodisconnect (timeout), share removed, or administrative permissions lacking.

Audit Connection Reject

```

struct ae_connrej {
    unsigned short ae_cr_compname; /* offset */
    unsigned short ae_cr_username; /* offset */
    unsigned short ae_cr_netname; /* offset */
    unsigned short ae_cr_reason;
};

```

where:

- *ae_cr_compname* is an offset (from the beginning address of the *ae_connrej* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_cr_username* is an offset (from the beginning address of the *ae_connrej* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_cr_username* and *ae_cr_compname* are the same.
- *ae_cr_netname* is an offset (from the beginning address of the *ae_connrej* data structure) to an ASCIIZ string indicating the desired netname of a resource.
- *ae_cr_reason* is one of four values indicating why the session was disconnected. These values are defined in the AUDIT.H header file, as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_USERLIMIT	0	Normal disconnection, or user name limit.
AE_BADPW	1	Error, session disconnect, or bad password.
AE_ADMINPRIVREQD	2	Autodisconnect (timeout), share removed, or administrative permissions lacking.
AE_NOACCESSPERM	3	No access permissions to shared resource.

Audit Resource Access Level 1

```

struct ae_resaccess {
    unsigned short ae_ra_compname; /* offset */
    unsigned short ae_ra_username; /* offset */
    unsigned short ae_ra_resname; /* offset */
    unsigned short ae_ra_operation;
    unsigned short ae_ra_returncode;
    unsigned short ae_ra_restype;
    unsigned short ae_ra_fileid;
};

```

where:

- *ae_ra_compname* is an offset (from the beginning address of the *ae_resaccess* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_ra_username* is an offset (from the beginning address of the *ae_resaccess* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_ra_username* and *ae_ra_compname* are the same.
- *ae_ra_resname* is an offset (from the beginning address of the *ae_resaccess* data structure) to an ASCIIZ string indicating the name of the resource accessed.

- *ae_ra_operation* is one of seven values indicating which operation was performed. These values, defined in the ACCESS.H header file, mean the following:

SYMBOLIC CONSTANT	BIT MASK	MEANING
ACCESS_READ	0x01	Data was read or run from a resource.
ACCESS_WRITE	0x02	Data was written to a resource.
ACCESS_CREATE	0x04	An instance of the resource (such as a file) was created; data may have been written to the resource while the resource was being created.
ACCESS_EXEC	0x08	A resource was run.
ACCESS_DELETE	0x10	A resource was deleted.
ACCESS_ATTRIB	0x20	Attributes of a resource were changed.
ACCESS_PERM	0x40	Permissions (read, write, create, execute, and delete) of a resource for a user or application were changed.

- *ae_ra_returncode* gives the return code from the particular operation. If 0, the operation was successful.
- *ae_ra_restype* gives the server message block (SMB) request function code.
- *ae_ra_fileid* gives the server identification number of a file.

Audit Resource Access Level 2

```
struct ae_resaccess2 {
    unsigned short ae_ra2_compname; /* offset */
    unsigned short ae_ra2_username; /* offset */
    unsigned short ae_ra2_resname; /* offset */
    unsigned short ae_ra2_operation;
    unsigned short ae_ra2_returncode;
    unsigned short ae_ra2_restype;
    unsigned long ae_ra2_fileid;
};
```

where:

- The first 6 fields in this data structure are identical to those in the previous level.
- *ae_ra2_fileid* gives the server identification number of a file. It is given as an unsigned long value, rather than as an unsigned short as in the *ae_resaccess* structure.

Audit Resource Access Rejected

```

struct ae_resaccessrej {
    unsigned short ae_rr_compname; /* offset */
    unsigned short ae_rr_username; /* offset */
    unsigned short ae_rr_resname; /* offset */
    unsigned short ae_rr_operation;
};

```

where:

- *ae_rr_compname* is an offset (from the beginning address of the *ae_resaccessrej* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_rr_username* is an offset (from the beginning address of the *ae_resaccessrej* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_rr_username* and *ae_rr_compname* are the same.
- *ae_rr_resname* is an offset (from the beginning address of the *ae_resaccessrej* data structure) to an ASCIIZ string indicating the name of the resource to which access was denied.
- *ae_rr_operation* is one of seven values indicating the operation requested. These values, defined in the ACCESS.H header file, mean the following:

SYMBOLIC CONSTANT	BIT MASK	MEANING
ACCESS_READ	0x01	Data was read or run from a resource.
ACCESS_WRITE	0x02	Data was written to a resource.
ACCESS_CREATE	0x04	An instance of the resource (such as a file) was created; data may have been written to the resource while the resource was being created.
ACCESS_EXEC	0x08	A resource was run.
ACCESS_DELETE	0x10	A resource was deleted.
ACCESS_ATTRIB	0x20	Attributes of a resource were changed.
ACCESS_PERM	0x40	Permissions (read, write, create, execute, and delete) of a resource for a user or application were changed.

Audit File Closed

```

struct ae_closefile {
    unsigned short ae_cf_compname; /* offset */
    unsigned short ae_cf_username; /* offset */
    unsigned short ae_cf_resname; /* offset */
    unsigned short ae_cf_fileid;
    unsigned long ae_cf_duration;
    unsigned short ae_cf_reason;
};

```

where:

- *ae_cf_compname* is an offset (from the beginning address of the *ae_closefile* data structure) to an ASCIIZ string indicating the

name of the workstation that established the session.

- *ae_cf_username* is an offset (from the beginning address of the *ae_closefile* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If this field is 0, the user name and workstation name are assumed to be the same.
- *ae_cf_resname* is an offset (from the beginning address of the *ae_closefile* data structure) to an ASCIIZ string indicating the name of the resource that owns the accessed files.
- *ae_cf_fileid* is the identification number of the file.
- *ae_cf_duration* specifies how many seconds the resource was used.
- *ae_cf_reason* gives the reason the session was disconnected. These values, defined in the AUDIT.H header file, mean the following:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_NORMALCLOSE	0	Normal client disconnection
AE_SES_CLOSE	1	Session disconnection
AE_ADMIN_CLOSE	2	Administrative disconnection

Audit Service Status or Text Change

An audit log entry is written when service-status auditing is on, and a service performs a NetServiceStatus call that updates the service status (*svcs_status*). An audit entry is written only if the status of one of the following values changes:

- INSTALLED
- UNINSTALLED
- PAUSED
- CONTINUED (ACTIVE)

```
struct ae_servicestat {
    unsigned short ae_ss_compname; /* offset */
    unsigned short ae_ss_username; /* offset */
    unsigned short ae_ss_svcname; /* offset */
    unsigned short ae_ss_status;
    unsigned long ae_ss_code;
    unsigned short ae_ss_text; /* offset */
    unsigned short ae_ss_returnval;
};
```

where:

- *ae_ss_compname* is an offset (from the beginning address of the *ae_servicestat* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_ss_username* is an offset (from the beginning address of the *ae_servicestat* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_ss_username* and *ae_ss_compname* are the same.
- *ae_ss_svcname* is an offset (from the beginning address of the *ae_servicestat* data structure) to an ASCIIZ string indicating the name of a service.
- *ae_ss_status* is the service status being set.
- *ae_ss_code* is the service code being set.
- *ae_ss_text* is an offset to text being set.
- *ae_ss_returnval* is the return value.

Audit Access Control List

An audit log entry is written when an existing access control profile record is changed or deleted.

```
struct ae_aclmod {
    unsigned short ae_am_compname; /* offset */
    unsigned short ae_am_username; /* offset */
    unsigned short ae_am_resname; /* offset */
    unsigned short ae_am_action;
    unsigned short ae_am_dataalen;
};
```

where:

- *ae_am_compname* is an offset (from the beginning address of the *ae_aclmod* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_am_username* is an offset (from the beginning address of the *ae_aclmod* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_am_username* and *ae_am_compname* are the same.
- *ae_am_resname* is an offset (from the beginning address of the *ae_aclmod* data structure) to an ASCIIZ string that indicates the name of a resource that owns the accessed files.
- *ae_am_action* is the action performed on the access control profile record, as follows:

VALUE	MEANING
0	Change
1	Deletion
2	Addition
9	Unsuccessful password change attempt (valid only for user record)

- *ae_am_dataalen* is the length of data following the fixed data structure. This is always 0 in records generated by LAN Server.

Audit User Account Subsystem

An audit log entry is written when (1) an existing user accounts subsystem (UAS) record is changed or deleted, or (2) the UAS modals are changed.

```
struct ae_uasmod {
    unsigned short ae_um_compname; /* offset */
    unsigned short ae_um_username; /* offset */
    unsigned short ae_um_resname; /* offset */
    unsigned short ae_um_rectype;
    unsigned short ae_um_action;
    unsigned short ae_um_dataalen;
};
```

where:

- *ae_um_compname* is an offset (from the beginning address of the *ae_uasmod* data structure) to an ASCIIZ string indicating the requester that established the session.

- *ae_um_username* is an offset (from the beginning address of the *ae_uasmod* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_um_username* and *ae_um_compname* are the same.
- *ae_um_resname* is an offset (from the beginning address of the *ae_uasmod* data structure) to an ASCIIZ string indicating the name of a resource that owns the accessed files.
- *ae_um_rectype* is the type of UAS record, as follows:

VALUE	MEANING
0	User record
1	Group record
2	UAS modals

- *ae_um_action* is the action performed on the UAS record, as follows:

VALUE	MEANING
0	Change
1	Deletion
2	Addition

- *ae_um_dataLEN* is the length of data following the fixed data structure. This is always 0 in records generated by LAN Server.

Audit User Network Logon Record

This record is written by the server that processes the network logon of the user.

```
struct ae_netlogon {
    unsigned short ae_no_compname; /* offset */
    unsigned short ae_no_username; /* offset */
    unsigned short ae_no_privilege;
    unsigned long ae_no_authflags;
};
```

where:

- *ae_no_compname* is an offset (from the beginning address of the *ae_netlogon* data structure) to an ASCIIZ string, indicating the requester that established the session.
- *ae_no_username* is an offset (from the beginning address of the *ae_netlogon* data structure) to an ASCIIZ string, indicating the name of the user who initiated the session. If 0, *ae_no_username* and *ae_no_compname* are the same.
- *ae_no_privilege* is the privilege of the user logging on, as follows:

SYMBOLIC CONSTANT	VALUE
AE_GUEST	0
AE_USER	1
AE_ADMIN	2

- *ae_no_authflags* is a reserved field.

Audit User Network Logoff Record

This record is written by the server that processes the network logoff of the user.

```
struct ae_netlogoff {
    unsigned short ae_nf_compname; /* offset */
    unsigned short ae_nf_username; /* offset */
    unsigned short ae_reserved1;
    unsigned short ae_reserved2;
};
```

where:

- *ae_nf_compname* is an offset (from the beginning address of the *ae_netlogoff* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_nf_username* is an offset (from the beginning address of the *ae_netlogoff* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_nf_username* and *ae_nf_compname* are the same.
- *ae_reserved1* is a reserved field. It must be set to 0.
- *ae_reserved2* is a reserved field. It must be set to 0.

Audit Network Logon Denied

An audit log entry is written when the network logon request is denied.

```
struct ae_netlogdenied {
    unsigned short ae_nd_compname; /* offset */
    unsigned short ae_nd_username; /* offset */
    unsigned short ae_nd_reason;
    unsigned short ae_nd_subreason;
};
```

where:

- *ae_nd_compname* is an offset (from the beginning address of the *ae_netlogdenied* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_nd_username* is an offset (from the beginning address of the *ae_netlogdenied* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_nd_username* and *ae_nd_compname* are the same.
- *ae_nd_reason* is the reason for logon denial, as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_GENERAL	0	General access denied.
AE_BADPW	1	Incorrect password.
AE_ACCRESTRICT	4	Forced off by account system because of account restriction, such as logon hours.
AE_UNDEFINEDUSER	9	User ID does not exist.

- *ae_nd_subreason* is the detail of the reason for denial. When *nd_reason* is AE_ACCRESTRICT, one of the following is true:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_LIM_UNKNOWN	0	Unknown or unavailable.
AE_LIM_LOGONHOURS	1	Logon hours.
AE_LIM_EXPIRED	2	Account expired.
AE_LIM_INVALID_WKSTA	3	Requester ID not valid.
AE_LIM_DISABLED	4	Account disabled.

Otherwise, this value is 0.

Audit Logon Limit Exceeded

An audit log entry is written when users remain logged on while their account limitations no longer permit them to be logged on.

```
struct ae_acclim {
    unsigned short ae_al_compname; /* offset */
    unsigned short ae_al_username; /* offset */
    unsigned short ae_al_resname; /* offset */
    unsigned short ae_al_limit;
};
```

where:

- *ae_al_compname* is an offset (from the beginning address of the *ae_acclim* data structure) to an ASCIIZ string indicating the requester that established the session.
- *ae_al_username* is an offset (from the beginning address of the *ae_acclim* data structure) to an ASCIIZ string indicating the name of the user who initiated the session. If 0, *ae_al_username* and *ae_al_compname* are the same.
- *ae_al_resname* is the offset to the resource name.
- *ae_al_limit* is the limit that was exceeded, as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
AE_LIM_UNKNOWN	0	Unknown or unavailable.
AE_LIM_LOGONHOURS	1	Logon hours.
AE_LIM_EXPIRED	2	Account expired.

Related Information

For information about:

- LAN Server network commands and the IBMLAN.INI file, see the *IBM Directory and Security Server Commands and Utilities* and the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.
- Permission levels, see [Access Permission Category](#) and [User Category](#).

NetAuditClear or Net32AuditClear

NetAuditClear or Net32AuditClear

The NetAuditClear API clears the audit log file and, optionally, saves the contents to another file.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <audit.h>

NetAuditClear(pszServername, pszBackupFile,
              reserved);    /* 16 bit */

Net32AuditClear(pszServername, pszBackupFile,
                reserved);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszBackupFile (const unsigned char LSFAR *) optionally points to an ASCIIZ string containing the name for a backup file. The calling application must have write privilege for the path specified by *pszBackupFile*. The path name also must be accessible by the OS/2 DosMove function. If the path name is relative, it is assumed to be relative to the IBMLAN\LOGS directory.

A NULL pointer tells NetAuditClear not to save the audit log entries.

reserved (unsigned char LSFAR *) must be NULL.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_FILENAME_EXCED_RANGE	206.	The file name is longer than 8 characters or the extension is longer than characters.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosDelete
- DosFsCtl
- DosGetShrSeg
- DosMove
- DosOpen

Remarks

The NetAuditWrite API (see [Auditing - NetAuditWrite](#)) issues an *admin* alert when the audit log file reaches 80% capacity and again when the file reaches 100% capacity. At 100%, NetAuditWrite fails. Therefore, applications should periodically clear the audit log file of outdated information.

To set a maximum size for the audit log file, use one of the following methods:

- Use the NET CONFIG command with the /MAXAUDITLOG option (see the *LAN Server Commands and Utilities* for more information).
- Set the *maxauditlog* parameter in the IBMLAN.INI file (see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning* for a description of IBMLAN.INI).
- Call the NetServerSetInfo API with the *sv_maxauditsz* parameter.

Related Information

For information about:

- Getting the status of audit log file capacity, see [Alert - NetAlertRaise](#).
- Writing an entry to the audit log file, see [Auditing - NetAuditWrite](#).

NetAuditRead or Net32AuditRead

NetAuditRead or Net32AuditRead

The NetAuditRead API reads from the audit log on a server.

The NetAuditRead API replaces the NetAuditOpen API, which is now obsolete.

To read an entire audit log, an application must call NetAuditRead repeatedly until the API indicates that there is no more data to be read. Each call to NetAuditRead returns a handle that must be provided to any subsequent call to NetAuditRead. This handle changes with each subsequent call; it is *not* a system file handle and never should be treated as such.

Note: The audit log can contain more than 64KB of data. If *pusBytesReturned* is 0 and *pusBytesAvail* is not 0, *usBuflen* is too small to hold all the records in the file, and the application should continue to reiterate calls to this API until *pusBytesReturned* is returned as 0.

The value of *pusBytesAvail* can be 0xFFFF. This value means there can be 0xFFFF or more bytes of data available, which possibly is even more than 64KB (the maximum buffer size). In order to receive all the data available from this API, an application must reiterate the call until *pusBytesReturned* and *pusBytesAvail* both are returned as 0.

pusBytesReturned applies to the last complete record in the buffer. However, a partial record may be left at the end of the buffer. You may not know whether the area following the last entry is null unless you check.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <audit.h>

NetAuditRead(pszServername, reserved1, pLogHandle, offset,
             reserved2, reserved3, flags, buf,
             usBuflen, pusBytesReturned, pusBytesAvail);    /* 16 bit */

Net32AuditRead(pszServername, reserved1, pLogHandle, offset,
               reserved2, reserved3, flags, buf,
               ulBuflen, pulBytesReturned, pulBytesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

reserved1 (const unsigned char LSFAR *) must be NULL.

pLogHandle (HLOG LSFAR *) is the pointer to the returned log handle.

An application calling NetAuditRead for the first time must initialize the 128-bit log handle as follows:

BITS	VALUE
127 (MSB) – 64	0
63 – 0 (LSB)	1

The most significant bit (MSB) is the leftmost bit, and the least significant (LSB) is the rightmost bit. To initialize, use the fields in the log-handle structure, which is defined in the AUDIT.H header file. The structure definition is:

```
typedef struct loghandle {
    unsigned long    time;
    unsigned long    last_flags;
    unsigned long    offset;
    unsigned long    rec_offset;
} HLOG;
```

Using the log-handle structure, you can initialize the bits in the following manner:

```
ploghndl.time = 0L;
ploghndl.last_flags = 0L;
ploghndl.offset = 0xFFFFFFFF;
ploghndl.rec_offset = 0xFFFFFFFF;
```

Thereafter, each call to NetAuditRead must be given the value for the log handle that was returned by the previous call to NetAuditRead.

offset (unsigned long) is the offset of the record to begin the read. This parameter is ignored unless the *flags* bit 1 is set. If the *flags* bit 1 is set, *offset* is taken as a zero-based offset based on the record number, not bytes, at which the data returned should begin. The *offset* parameter is dependent on which direction a file is read. If the file is being read backward, record 0 is the last record in the file. If the file is being read forward, record 0 is the first record in the file.

reserved2 (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) must be NULL.

reserved3 (unsigned long) must be 0.

flags (unsigned long) is a bit mask, mapped as follows:

BITS	MEANING
0	If 0, the file is read normally. If 1, the file is read in reverse chronological order and records are returned in the buffer in reverse chronological order (last records first).
1	If 0, read proceeds normally and sequentially. If 1, read proceeds from the nth record from the start of the file, where n is the offset parameter.
2-31	Reserved; must be 0.

buf (unsigned char LSFAR *) points to the local memory address where the audit log file entries are to be placed.

usBufLen (16-bit unsigned short or 32-bit unsigned long) specifies the amount of local memory (in bytes) allocated to receive data from this API.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_NET_WRITE_FAULT	88	A network data fault has occurred.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_LogFileChanged	2378	This log file has changed between reads.
NERR_LogFileCorrupt	2379	This log file is damaged.
NERR_InvalidLogSeek	2440	The log file does not contain the requested record number.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosClose
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead

Related Information

For information about:

- Clearing an audit log file, see [Auditing - NetAuditClear](#).
- Closing an audit log file, see DosClose in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Reading an audit log file, see DosRead in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Writing an entry to the audit log file, see [Auditing - NetAuditWrite](#).

NetAuditWrite or Net32AuditWrite

NetAuditWrite or Net32AuditWrite

The NetAuditWrite API writes an audit trail entry to the local audit log file.

Restrictions

A call to this API can be issued only from an OS/2 application to a local LAN Server workstation.

Syntax

```
#include <netcons.h>
#include <audit.h>

NetAuditWrite(audtype, buf, usBuflen,
              reserved1, reserved2);    /* 16 bit */

Net32AuditWrite(audtype, buf, ulBuflen,
                reserved1, reserved2);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- audtype* (16-bit unsigned short or 32-bit unsigned long) specifies the type of entry to write to the audit log file. This parameter is the same as the *ae_type* field, described in [Audit Entry Header](#).
- buf* (const unsigned char LSFAR *) points to the variable-length data in the data structure, as described in [Audit Entry Data Structures](#). (*buf* does not include the fixed-length header portion of the audit_entry structure.)
- reserved1* (unsigned char LSFAR *) must be NULL.
- reserved2* (unsigned char LSFAR *) must be NULL.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_NET_WRITE_FAULT	88	A network data fault has occurred.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NERR_LogOverflow

2377

This log file exceeds the maximum defined size.

Remarks

NetAuditWrite issues an *admin* alert by calling the NetAlertRaise API when the audit log file reaches 80% capacity and again when the file reaches 100% capacity. At 100% audit log capacity, the NetAuditWrite API fails, returning the error code NERR_LogOverflow.

To return successfully, the NetAuditWrite API requires that the auditing entry in the IBMLAN.INI file be set to YES, as follows:

```
AUDITING = YES
```

Related Information

For information about:

- Closing the audit log file, see DosClose in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Retrieving the size of the audit log file, see [Server - NetServerGetInfo](#).
- Setting the maximum size of the audit log file, see [Server - NetServerSetInfo](#).

Configuration Category

This category includes the following APIs:

[Configuration - NetConfigGet2](#)
[Configuration - NetConfigGetAll2](#)

Configuration APIs retrieve network configuration information from the IBMLAN.INI file. The NetConfigGet2 API retrieves a single parameter value for a given network component; NetConfigGetAll2 returns all the parameters for the given component. These APIs are used with the CONFIG.H and NETCONS.H header files.

The IBMLAN.INI file is an ASCII file containing the configuration information for LAN Server services. User-defined services and applications also store network configuration information in this file.

The IBMLAN.INI file consists of component lines, parameter lines, and comment lines in a format that enables the configuration functions to browse through and retrieve the information. The format is as follows:

- Component lines mark the start of information about a component, in the form:

[componentname]

- Parameter lines contain a parameter and a value, in the form:

parameter = value

The parameter value can consist of arbitrary text and is not processed by the configuration APIs, except that leading and trailing spaces are stripped. Interpretation of the value is left to the caller. No quotation marks are allowed as part of the parameter value.

If a parameter appears several times for any one component, NetConfigGetAll2 returns each occurrence; NetConfigGet2 returns only the first occurrence. The same parameter name can be used under different components without affecting the NetConfigGet2 return.

- Comment lines are any blank lines or lines in which the first nonblank character is a semicolon (;).

An IBMLAN.INI requester component might contain the following information:

```
[requester]
; define net_tool requester
computername = net_tool
charcount = 16
```

As shown, the *requester* defines a computer name of *net_tool* and specifies that 16 bytes of characters must accumulate before a requester sends them to a serial device queue.

Note: The IBMLAN.INI file contains default values for network components. These values might not reflect actual values passed to a network service. If you notice inconsistencies between IBMLAN.INI entries and actual service values, examine the manner in which you have defined these service values within your application. To maintain consistency, call NetConfigGetAll2 and examine the returned values before setting new parameter values for a given component.

DOS Considerations

Configuration APIs can be called locally on a DLS requester to retrieve information from the local NETWORK.INI file. These APIs can be called remotely from a DLS requester to retrieve information from a remote IBMLAN.INI file.

Related Information

For information about:

- The IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.
- The NETWORK.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetConfigGet2 or Net32ConfigGet2

NetConfigGet2 or Net32ConfigGet2

The NetConfigGet2 API retrieves one specified parameter value from the IBMLAN.INI file of the local computer or a remote server.

The NetConfigGet2 API replaces the NetConfigGet API, which is now obsolete.

NetConfigGet2 returns the value (ASCII string) for a single parameter of a specified component in *but*. This string is the entire value content of the IBMLAN.INI line for the specified parameter, which is all text to the right of the equals sign (=). Leading and trailing spaces are stripped from this text. No other processing is performed on it.

Restrictions

This API can be called from DLS and OS/2 workstations. (A local call on a DLS workstation retrieves information from the local NETWORK.INI file.) Administrative authority is required to call this API at a remote server.

Syntax

```
#include <netcons.h>
#include <config.h>

NetConfigGet2(pszServername, reserved, component,
```

```

        parameter, buf, usBuflen,
        pulBytesReturned);          /* 16 bit */

Net32ConfigGet2(pszServername, reserved, component,
        parameter, buf, ulBuflen,
        pulBytesAvail);             /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

reserved (const unsigned char LSFAR *) must be NULL.

component (const unsigned char LSFAR *) points to an ASCIIZ string specifying the name of the component to be searched.

parameter (const unsigned char LSFAR *) points to an ASCIIZ string specifying the parameter for which a value is to be returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_CfgCompNotFound	2146	The program could not find the specified component in the IBMLAN.INI file.

NERR_CfgParamNotFound	2147	The program could not find the specified parameter in the IBMLAN.INI file.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions :

- DosChgFilePtr
- DosDuphandle
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead

NetConfigGetAll2 or Net32ConfigGetAll2

NetConfigGetAll2 or Net32ConfigGetAll2

The NetConfigGetAll2 API retrieves all configuration information for a given network component in the IBMLAN.INI file of a local computer or a remote server.

The NetConfigGetAll2 API replaces the NetConfigGetAll API, which is now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations. (A local call on a DLS workstation retrieves information from the local NETWORK.INI file.) Administrative authority is required to call this API at a remote server.

Syntax

```
#include <netcons.h>
#include <config.h>
NetConfigGetAll2(pszServername, reserved, component,
                buf, usBuflen, pulBytesReturned,
                pulBytesAvail);          /* 16 bit */

Net32ConfigGetAll2(pszServername, reserved, component,
                  buf, ulBuflen, pulBytesReturned,
                  pulBytesAvail);      /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>reserved</i>	(const unsigned char LSFAR *) must be NULL.
<i>component</i>	(const unsigned char LSFAR *) points to an ASCIIZ string specifying the name of the component to search.
<i>buf</i>	(unsigned char LSFAR *) points to the memory address where the parameter values of a component are to be returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_CfgCompNotFound	2146	The program could not find the specified component in the IBMLAN.INI file.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosDuphandle
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead

Remarks

NetConfigGetAll2 returns in *buf* a set of concatenated ASCIIZ strings that represent configuration information for the specified component. Each string is ended by NULL (ASCII 0), and the whole buffer is ended by a null string. Information is returned in the form *parm=value*. The parameter name, which appears to the left of the equals sign (=), is in uppercase. The *pusBytesReturned* and *pusBytesAvail* values are filled in as they are for GetInfo calls.

For example,

```
" foo = Bar,1,long comment string "
```

in the IBMLAN.INI file is returned as:

```
"FOO=Bar,1,long comment string"
```

Connection Category

This category includes the following API:

[Connection - NetConnectionEnum](#)

The NetConnectionEnum API lists all connections made to a server by a requester client, or all connections made to a shared resource of a server.

A requester accesses a shared resource of a server by means of a connection. Thus, a *connection* is the path between a redirected local device name of a requester and a shared resource of a server. Using a NetUseAdd (UNC) name can establish a connection without any local device name.

This API is used with the SHARES.H and NETCONS.H header files.

Connection Data Structures

The NetConnectionEnum API returns data at a detail *sLevel* of 0 or 1, using the following data structures.

Connection Level 0

```
struct connection_info_0 {
    unsigned short conil0_id;
};
```

where *conil0_id* is the connection identification number.

Connection Level 1

```
struct connection_info_1 {
    unsigned short          conil_id;
    unsigned short          conil_type;
    unsigned short          conil_num_opens;
    unsigned short          conil_num_users;
    unsigned long           conil_time;
    unsigned char LSFAR * LSPTR conil_username;
    unsigned char LSFAR * LSPTR conil_netname;
};
```

where:

- The first field in this data structure is identical to that in the previous level.

- *con1_type* indicates the type of connection made from the local device name to the shared resource. The SHARES.H header file defines the following types of connection:

SYMBOLIC CONSTANT	VALUE	MEANING
STYPE_DISKTREE	0	Disk connection
STYPE_PRINTQ	1	Spooler queue connection
STYPE_DEVICE	2	Serial device connection
STYPE_IPC	3	Interprocess communication (IPC) connection

- *con1_num_opens* indicates the number of files that currently are open as a result of the connection.
- *con1_num_users* indicates the number of users on the connection.
- *con1_time* indicates the number of seconds the connection has been established.
- *con1_username* points to an ASCIIZ string indicating the user that made the connection.
- *con1_netname* points to an ASCIIZ string indicating (1) the netname of the shared resource of the server, or (2) the computer name of the requester, depending on which name was specified as the *qualifier* parameter of the NetConnectionEnum API. The type of name supplied to *con1_netname* is the inverse of the type supplied to the *qualifier* parameter.

Related Information

For information about connecting a device name of a requester to a shared resource of a server, see [Use - NetUseAdd](#).

NetConnectionEnum or Net32ConnectionEnum

NetConnectionEnum or Net32ConnectionEnum

The NetConnectionEnum API lists (1) the connections made to a shared resource of a server or (2) all connections established from a particular computer to a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. A call to this API can be issued only to an OS/2 LAN Server workstation.

Syntax

```
#include <netcons.h>
#include <shares.h>

NetConnectionEnum(pszServername, qualifier, sLevel, buf,
                 usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32ConnectionEnum(pszServername, qualifier, ulLevel, buf,
                   ulBuflen, pulEntriesReturned, pulEntriesAvail);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

qualifier (const unsigned char LSFAR *) points to an ASCIIZ string specifying either (1) the netname of the shared resource whose connections are to be listed, or (2) the client name of the requester whose connections to the shared resource are to be listed. This value cannot be a null pointer or string.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Connection Level 0](#) and [Connection Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_ClientNameNotFound	2312	A session does not exist with that computer name.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

If *qualifier* specifies a requester, NetConnectionEnum returns a list of all connections made between the requester and the specified server during the current session.

If *qualifier* specifies a shared resource, NetConnectionEnum returns a list of all connections made to the shared resource.

Related Information

For information about:

- Listing all available servers, see [Server - NetServerEnum2](#).
- Listing sessions on a server, see [Session - NetSessionEnum](#).

DLS-Specific APIs

This category includes the following APIs:

- [DLS - Installation Check](#)
- [DLS - Network Print Stream Control](#)
- [DLS - Get User ID and Logon Status](#)
- [DLS - DLS Installation Check](#)
- [DLS - Network Version Check](#)
- [DLS - NetWkstaSetUID2](#)

DOS LAN Services (DLS) provides several function calls for compatibility with applications currently supported by both DLS and PC LAN Program 1.3. This section contains an overview and descriptions of those function calls.

NOTE TO DOUBLE-BYTE CHARACTER SET (DBCS) USERS: DLS runs on hardware that supports double-byte characters. The interrupt calls described here for DLS return DLS-specific information enabling you to develop applications that run on DBCS hardware.

For more information about DLS support and an example of how to use DosReadAysncNmPipe and DosWriteAysncNmPipe, refer to [Notes for DOS Applications](#).

Function Call Overview

DLS supports the following DOS interrupt function calls.

INTERRUPT	FUNCTION	DESCRIPTION
2AH	0000	Installation check
2AH	060x	Network print stream control
2AH	7802	Get user ID and logon status
2FH	B800	DLS installation check
2FH	B809	Network version check

0000H (INT 2AH) Installation Check

0000H (INT 2AH) Installation Check

This API checks to see if the interrupt 2AH interface has been installed. A program can verify whether the interrupt 2AH interface is loaded by calling interrupt 2AH Installation Check.

ON ENTRY	REGISTER CONTENTS
AH	0
ON RETURN	REGISTER CONTENTS
AH	Installed Flag
	"Zero ; Not installed"
	"Non-Zero ; Installed"

060xH (INT 2AH) Network Print Stream Control

060xH (INT 2AH) Network Print Stream Control

This API controls the truncation and concatenation of print streams for network printers.

ON ENTRY	REGISTER CONTENTS
AH	06
AL	01 ; Set print output into Concatenation mode
	02 ; Set print output into Truncation mode
	03 ; Truncate print stream
ON RETURN	REGISTER CONTENTS
AX	DOS error code if carry set None if carry flag not set

Remarks

DOS defines separate print streams for each of up to nine redirected printers. These printers are LPT1 (or PRN)-LPT9.

Note: PC LAN Program had only one print stream for each redirected print device. DLS has one interrupt 17H print stream for each redirected print device and multiple interrupt 21H print streams for each redirected print device.

Interrupt 17H print streams are delimited by the following events:

- End of program
- Printing with interrupt 17H from a different DOS process (created by the DOS EXEC function)
- Opening and closing of the file LPT x (1-9) without sending any data to it

DOS interrupt 21H printing is delimited by the following events:

- Closing the file LPT x (1-9)
- Ending the DOS process that opened the file LPT x (1-9)

DOS normally is in Truncation mode, which means the interrupt 17H print stream delimiters previously listed take effect. Concatenation mode (AL=01) causes DOS to ignore the interrupt 17H stream delimiters. In this mode, the interrupt 17H print stream is delimited only when a single DOS command ends or when a DOS batch (.BAT) process ends. This allows output from several commands to be kept together if the commands are in a batch file. Set Concatenation mode changes the stream state for all redirected printers. It has no effect on nonredirected printers or file handle printing.

Set Truncation mode (AL=02) returns DOS to Truncation mode. It is used to cancel the Set Concatenation mode function. This call does not truncate streams; the next stream delimiter causes the truncation. Truncation mode is set automatically at the end of all DOS batch files and at the end of any command not in a batch file. Set Truncation mode changes the stream state for all redirected printers. It has no effect on nonredirected printers.

Not all programs print output in such a way that DOS can determine when a print stream ends. This situation occurs most often when interrupt 17H is used for printing. Truncate Print Stream (AL=03) allows a program to indicate that the data currently printed is a complete stream and should be ended and printed.

This function serves the same purpose as the Ctrl+Alt+* (on the numeric keypad) key sequence that is available to users to end print streams. Truncate Print Stream truncates all interrupt 17H print streams. It has no effect on nonredirected printers or file handle printing. The DLS program must be started for this function call to work.

7802H (INT 2AH) Get User ID and Logon Status

7802H (INT 2AH) Get User ID and Logon Status

This API provides the user ID and logon status information for the current user of the DLS program.

ON ENTRY	REGISTER CONTENTS
AX	7802H
ES:DI	Address of 8 byte buffer
ON RETURN	REGISTER CONTENTS
AL	"Zero ; No user logged on" "Non-zero ; User logged on"
ES:DI	Buffer contains user ID, ASCII, padded with blanks.

B800H (INT 2FH) DLS Installation Check

B800H (INT 2FH) DLS Installation Check

This API checks to see if DLS has been installed. If the program is installed, Installation Check returns the installed network components.

To determine which configuration the users specified when they started the network, perform the component checks in the order indicated in the preceding assembler code. This order is required because a given configuration might install more than one component.

```
ON      REGISTER
ENTRY   CONTENTS

AX      B800H


ON      REGISTER
RETURN  CONTENTS

AL      Network installed flag:

        "Zero      ; Network not installed"
        "Non-zero ; Network is installed"

BX      Installed component flag (bit flags):

        "xxxxxxxx xxxlxxx : Redirector"
        "xxxxxxxx lxxxxxxx : Receiver"
```

NOTE: x represents an undefined value.

Assembler Usage

```
MOV     AX,B800H           ;AX - function code
INT      2FH               ;Call function
CMP     AL,0               ;Is network installed?
JE      NOT_INSTALLED      ;No, network not installed.
TEST    BX,RECEIVER_FLAG   ;Is Receiver installed?
JNZ     RECEIVER_STARTED   ;Yes, Receiver is installed.
TEST    BX,REDIRECTOR_FLAG ;Is Redirector installed?
JNZ     REDIRECTOR_STARTED ;Yes, Redirector is installed.
```

B809H (INT 2FH) Network Version Check

B809H (INT 2FH) Network Version Check

This API returns the version level of DLS.

```
ON      REGISTER
ENTRY   CONTENTS

AX      B809H
```

ON	REGISTER
RETURN	CONTENTS
AH	Minor version number
AL	Major version number

Remarks

First perform a network installation check (AX=B800H) to determine if the network has been installed. The Network Version Check function returns the hexadecimal equivalent of the DLS version number. For DLS versions prior to 2.0, '1.40' (AH=28H, AL=01H) is returned, rather than '1.20' or '1.30', to avoid confusion with the identical PC LAN Program version.

NetWkstaSetUID2

NetWkstaSetUID2

The NetWkstaSetUID2 API performs the following functions:

- Logs a user name on to a requester
- Logs a user name off of a requester
- Runs a logon script on a logon server for a user name

For users at DLS workstations, the NetWkstaSetUID2 API is only part of the logon processing. When using the NetWkstaSetUID2 API with DLS workstations, remember that:

- User Profile Management (UPM) does not recognize logons or logoffs from this API. UPM is for DLS local requester logon only.
- NetWkstaSetUID2 does not provide for connection of logon assignments defined for the user ID in the DLS or LAN Requester graphical user interface.

This function takes an optional domain name argument. If that argument is NULL, the API defaults to the primary domain of the requester, specified in the NETWORK.INI file for a DLS workstation.

The NetWkstaSetUID2 API replaces the NetWkstaSetUID API, which is now obsolete.

Syntax

```
#include <netcons.h>
#include <wksta.h>
#include <access.h>

NetWkstaSetUID2(reserved, pszDomainName, pszUserID,
               pszPassword, parms, ucond, sLevel,
               buf, usBuflen, pusBytesAvail);
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>reserved</i>	(unsigned char LSFAR *) must be NULL.
<i>pszDomainName</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of the domain to log on to. If this parameter is NULL or points to a null string, the primary domain of the requester is

used.

pszUserID (unsigned char LSFAR *) points to an ASCIIZ string containing the user name to be logged on. Specifying a null string logs off the user name that is presently logged on to the requester.

pszPassword (unsigned char LSFAR *) points to an ASCIIZ string containing the password of the user name, obtained by an application's request to the user. A null pointer or string indicates that no password is needed. After a user is logged on to a requester, the *pszPassword* and *pszUserID* are verified again by LAN Server whenever the same requester attempts to access a remote resource.

parms (unsigned char LSFAR *) is reserved and must be NULL.

ucond (unsigned short) specifies what action to take if another user name is logged on to the requester. The WKSTA.H header file defines four values:

SYMBOLIC CONSTANT	VALUE	MEANING
WKSTA_NOFORCE	0	NetWkstaSetUID2 fails, and the user session number (UID) does not change.
WKSTA_FORCE	1	Logs the current user name off, disconnects any connections to redirected resources.
WKSTA_LOTS_OF_FORCE	2	Cancels any connections and other activities necessary. Fails if another user is used by a process as the current user.
WKSTA_MAX_FORCE	3	Always succeeds; forces all disconnected resources.

sLevel (unsigned short) must be 1. Depending on the *pszUserID* submitted, *sLevel* specifies either the *user_logon_info_1* or the *user_logoff_info_1* data structure, described in [User Logon Level 1](#) and [User Logoff Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. See [API Return Codes](#) for a complete list of return codes.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_AlreadyLoggedIn	2200	This workstation is already logged on.
NERR_NotLoggedIn	2201	This workstation has not been logged on yet.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.

NERR_BadPassword	2203	The specified password is not valid.
NERR_UnableToAddName_W	2204	The logon processor did not add the message alias.
NERR_UnableToAddName_F	2205	The logon processor did not add the message alias.
NERR_UnableToDelName_W	2206	The logoff processor did not delete the message alias.
NERR_UnableToDelName_F	2207	The logoff processor did not delete the message alias.
NERR_ActiveConns	2402	Active connections still exist.
NERR_PausedConns	2406	Paused connections could not be deleted.
NERR_LastAdmin	2452	The last administrator cannot be deleted.
NERR_LogonTrackingError	2454	Unable to set logon information for this user.
NERR_NetLogonNotStarted	2455	The Netlogon service has not been started.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosSemClear
- DosWrite
- int2F[2F_NetSetUserName]

Remarks

If a user name already is logged on to the requester (UID is already in effect), NetWkstaSetUID2 takes the action specified in *ucond*, either failing or forcing the user to log off.

If the user's logon is not valid, the API returns ERROR_ACCESS_DENIED. The *usrlog1_code* field in the *user_logon_info_1* data structure is valid even when the API returns ERROR_ACCESS_DENIED. The other fields are valid only when the return code is Validated Logon.

The values of the *usrlog1code* fields can be as follows:

SYMBOLIC CONSTANT	MEANING
NERR_Success	No errors were encountered.
NERR_PasswordExpired	The user has an account, but the user's password has expired. No other conditions of logon have been checked.

NERR_InvalidWorkstation	The user was attempting to log on from a requester that is not valid.
NERR_InvalidLogonHours	The user was attempting to log on at a time that is not valid.
ERROR_ACCESS_DENIED	Some condition of logon has not been met.
NERR_Stand-aloneLogon	No domain controller could be found to validate the user. Script processing was not performed.
NERR_NonValidatedLogon	The logon server could not validate the logon request.
NERR_LogonScriptError	An error occurred while processing the logon script.

The following table defines the other fields that are valid in the *user_logon_info_1* data structure for each code listed in the previous table:

RETURN CODE	MANIFEST LOGON	VALID FIELDS
NERR_Success	NERR_Stand-alone	None
NERR_UnableToAddName_W	NERR_NonValidatedLogon	Computer name, Script path
	NERR_Success	All
ERROR_ACCESS_DENIED	NERR_PasswordExpired	None
	NERR_InvalidWorkstation	None
	NERR_InvalidLogonHours	None
	NERR_LogonScriptError	None
	ERROR_ACCESS_DENIED(1)	None
Other Return Codes	None. Code is meaningless	None

(1) Because no account exists, the account is disabled, the account has expired, or the password does not match.

The following table defines fields for logging off:

RETURN CODE	MANIFEST LOGON	VALID FIELDS
NERR_Success	NERR_Stand-aloneLogon	None
NERR_UnableToDelName_W	NERR_NonValidatedLogon	None
	NERR_Success	All
All other errors	None. Code is meaningless	None

The *ucond* parameter is pertinent only when the API is called to log someone off. The following are its possible

meanings:

CHECK ALL OUTSTANDING CONDITIONS	UCOND=0	UCOND=1	UCOND=2	UCOND=3
1) Any drive current	Fail. Do no disconnects.	Fail. Do no disconnects.	Fail. Do no disconnects.	Force dis- connect, succeed.
2) Any drive	Fail. Do no disconnects.	Fail. Do no disconnects.	Force closed, succeed.	Force closed, succeed.
3) Any in use (NET USE...)	Fail. Do no disconnects.	Disconnect, unuse, succeed.	Disconnect, unuse, succeed.	Disconnect, unuse, succeed.
4) Anything dormant	Disconnect and succeed.	Disconnect and succeed.	Disconnect and succeed.	Disconnect and succeed.

The NetWkstaSetUID2 API is only part of the logon processing. When using the NetWkstaSetUID2 API with DLS workstations, keep the following in mind:

- NetWkstaSetUID2 does not provide for the connection of logon assignments defined for the user ID in the DLS or OS/2 Requester graphical user interface.
- A NetWkstaSetUID2 logon is not recognized by the DLS NET commands, and commands that require DOS LAN Requester logon will be unsuccessful.
- Any attempt to log on to a DLS workstation using the NET LOGON command or LAN Requester graphical user interface following a NetWkstaSetUID2 logon logs off the user that is logged on by the API.

Related Information

For information on configuring the local requester, see [Requester - NetWkstaSetInfo](#).

Error Logging Category

This category includes the following APIs:

[Error Logging - NetErrorLogClear](#)
[Error Logging - NetErrorLogRead](#)
[Error Logging - NetErrorLogWrite](#)

Error logging APIs control the error log file. The APIs are used with the ERRLOG.H and NETCONS.H header files.

Each time an error occurs during a network operation, an error log entry can be generated by NetErrorLogWrite to record the error information. The other two functions enable opening (NetErrorLogRead) and clearing (NetErrorLogClear) of the error log file, which stores the entries.

The error log file contains information about the following types of errors:

- LAN Server software internal errors
- OS/2 internal errors
- Network service errors

Error logs are stored in a combination of binary and ASCII characters. The default error log file name is \IBMLAN\LOGS\NET.ERR. All error logging functions perform their operations on this file.

DOS Considerations

Under DOS, the NetErrorLogClear and NetErrorLogRead APIs can be run only on a remote server. The NetErrorLogWrite API can be run only on a local requester. A DLS peer workstation does not support auditing or error logging.

Error Logging Data Structures

The NetErrorLogWrite and Net32ErrorLogWrite APIs return entries in the format of the *error_log* data structure when the error log file is read. The entry consists of a fixed-length data structure optionally followed by (1) 0 or more ASCIIZ strings (*el_text*) describing the error message, and (2) a block of raw data (*el_data*) relating to the cause of the error. Because of the variable lengths and structures of the *el_data* and *el_text* portions of the entry, only the fixed-length data structure is defined in the *error_log* data structure.

The fixed portion of the error log entry has the following format:

```
struct error_log {
    unsigned short    el_len;
    unsigned short    el_reserved;
    unsigned long     el_time;
    unsigned short    el_error;
    unsigned char     el_name[SNLEN+1];
    unsigned short    el_data_offset; /* offset from beginning
                                     address of error_log */
    unsigned short    el_nstrings;
};
/* variable-length data specific to the error
 * message and block of data associated with error */

unsigned char el_text [];          /* error message */
unsigned char el_data [];

/* raw data - the number of bytes used for raw data is equivalent to:
 * size = el_len - (el_data_offset + sizeof(el_len) ); */
unsigned short el_len;
```

where:

- *el_len* indicates the length (in bytes) of the error log entry. (Note that *el_len* is included at both the beginning and end of the entry to enable both forward and backward scanning of the file.)
- *el_reserved* is reserved.
- *el_time* indicates the time when *el_name* submitted the error entry.
- *el_error* is the error code for the error. This field can be used to obtain an error message from the NET.MSG file.
- *el_name* is an ASCIIZ string indicating the name of the network service or application that returned the error entry.
- *el_data_offset* specifies the byte offset from the beginning of the error log entry to the start of its variable-length portion (*el_data*).
- *el_nstrings* indicates the number of ASCIIZ strings the *el_text* portion of the entry contains.
- *el_text* points to 0 or more ASCIIZ strings describing the error. These ASCIIZ strings directly follow each *error_log* structure in the buffer returned by the NetErrorLogRead or Net32ErrorLogRead APIs.
- *el_data* points to the raw data associated with the error.

Related Information

For information about return codes, see [API Return Codes](#).

NetErrorLogClear or Net32ErrorLogClear

NetErrorLogClear or Net32ErrorLogClear

The NetErrorLogClear API clears (and optionally saves) the error log file of a computer.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <errlog.h>

NetErrorLogClear(pszServername, pszBackupFile,
                reserved);          /* 16 bit */

Net32ErrorLogClear(pszServername, pszBackupFile,
                  reserved);      /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- pszBackupFile* (const unsigned char LSFAR *) points to an ASCIIZ string assigning a name for an optional backup file. The calling application must have write privileges for the path specified by this parameter. In addition, the path name must be accessible by the OS/2 DosMove function. If the path name is relative, it is assumed to be relative to the IBMLAN\LOGS directory.

A null pointer indicates that NetErrorLogClear is not to save the error log entries.
- reserved* (unsigned char LSFAR *) is a null pointer.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_FILENAME_EXCED_RANGE	206.	The file name is longer than 8 characters or the extension is longer than characters.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosDelete
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosMove
- DosOpen

Remarks

NetErrorLogClear fails if another process currently is using the error log file.

The NetErrorLogWrite function issues one *admin* alert when the error log file reaches 80% capacity and another when the file reaches 100% capacity. At 100% error log file capacity, NetErrorLogWrite fails. Therefore, applications periodically should clear the error log file of outdated information.

To set a maximum size for the error log file, use one of the following methods:

- Use the NET CONFIG command with the /MAXERRORLOG option to override the option for a logon session, or edit the IBMLAN.INI file to change the value of the capacity parameter that determines the size of the error log. (For more information, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.)
- Set the *MAXERRORLOG* parameter in the IBMLAN.INI file. (For a description of the IBMLAN.INI file, see the *IBM Directory and Security Server Commands and Utilities*.)
- Call the NetWkstaSetInfo function with the *wki0_errlogsz* parameter.

Related Information

For information about writing an entry to the error log file, see [Error Logging - NetErrorLogWrite](#).

NetErrorLogRead or Net32ErrorLogRead

NetErrorLogRead or Net32ErrorLogRead

The NetErrorLogRead API reads from the specified computer error log. The NetErrorLogRead API replaces the NetErrorLogOpen API, which is now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required.

Syntax

```
#include <netcons.h>
#include <errlog.h>

NetErrorLogRead(pszServername, reserved1, pLogHandle,
               offset, reserved2, reserved3,
               flags, buf, usBuflen,
               pusBytesReturned, pusBytesAvail);           /* 16 bit */

Net32ErrorLogRead(pszServername, reserved1, pLogHandle,
                 offset, reserved2, reserved3,
                 flags, buf, ulBuflen,
                 pulBytesReturned, pulBytesAvail);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

reserved1 (const unsigned char LSFAR *) must be a null pointer.

pLogHandle (HLOG LSFAR *) points to the returned log handle. An application calling NetErrorLogRead for the first time must initialize the 64-bit log handle as follows:

BITS	VALUE
127 (MSB)–64	0
63–0 (LSB)	1

where the least significant bit (LSB) is the last (rightmost) bit.

offset (unsigned long) begins the read. This parameter is ignored unless *flags* bit 1 is set. If this bit is set, *offset* is taken as a zero-based offset based on the record number, rather than bytes, at which the data returned should begin.

Note: The record offset parameter is zero-based from both directions, depending on the direction of the read. If reading backward is specified, then record 0 is the last record in the file. If reading forward is specified, then record 0 is the first record in the file.

reserved2 (unsigned short LSFAR *) must be a null pointer.

reserved3 (unsigned long) must be 0.

flags (unsigned long) specifies the open flags. The bitmapped *flags* fields are:

BITS	MEANING
0	If 0, the file is read normally. If 1, the file is read in reverse chronological order (records first).

1 If 0, read proceeds normally and sequentially. If 1, from the nth record from the start of the file. n is parameter.

2-31 Reserved; must be 0.

buf (unsigned char LSFAR *) is the pointer to the buffer for returned data. The data includes 0 or more entries. The format of each entry is described under [Error Logging Data Structures](#) at the beginning of this chapter.

Note: If *pusBytesReturned* is 0 and *pusBytesAvail* is not 0, the buffer is too small to hold the next record in the file.

The error log can contain much more than 64KB of data. If *pusBytesAvail* is returned with a value of 0xFFFF, there can be 0xFFFF or more bytes of data available. The application should continue to read the log until the value returned in *pusBytesAvail* is 0.

pusBytesReturned applies to the last complete record in the buffer. However, a partial record may be left at the end of the buffer. You might not know whether the area following the last entry is null unless you check.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

NERR_LogFileChanged	2378	This log file has changed between reads.
NERR_LogFileCorrupt	2379	This log file is damaged.
NERR_InvalidLogSeek	2440	The log file does not contain the requested record number.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosClose
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead

Remarks

To read the contents of an error log, an application calls `NetErrorLogRead` repeatedly until the function indicates (through the *pusBytesAvail* parameter) that there is no more data to be read. Each call to `NetErrorLogRead` returns a handle that must be provided to any subsequent read call. This handle changes with each subsequent call. The handle is *not* a system file handle and should never be treated as such.

The 64-bit log handle must be initialized by using the fields in the structure defined in `ERRLOG.H`. The structure definition is:

```
typedef struct loghandle {
    unsigned long    time;
    unsigned long    last_flags;
    unsigned long    offset;
    unsigned long    rec_offset;
} HLOG;
```

Using the log-handle structure, the bits can be initialized in the following manner:

```
ploghdl.time = 0L;
ploghdl.last_flags = 0L;
ploghdl.offset = 0xFFFFFFFF;
ploghdl.rec_offset = 0xFFFFFFFF;
```

Thereafter, each call to `NetErrorLogRead` must be given the value for the log handle that was returned by the previous call to `NetErrorLogRead`. The data is returned in the buffer. The application should use the *pusBytesReturned* value to determine the end of valid data in the buffer.

Any application calling either the `NetErrorLogRead` or `Net32ErrorLogRead` API uses the `DosGetMessage` API to retrieve the message text corresponding to the *el_error* value from the appropriate message file. Most entries in the error log resolve to either the `NET.MSG` (for LAN Server) or `OSO001.MSG` (for OS/2) files, although applications can log errors that resolve to other message files.

Related Information

For information about:

- Clearing an error log file, see [Error Logging - NetErrorLogClear](#).
- Closing an error log file, see `DosClose` in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Reading an error log file, see `DosRead` in the *OS/2 Technical Library, Programming Guide Volume 1*.

NetErrorLogWrite or Net32ErrorLogWrite

NetErrorLogWrite or Net32ErrorLogWrite

The NetErrorLogWrite (local) API writes an entry to the error log file. Using the OS/2 program, the entry goes in the NET.ERR file.

Restrictions

A call to this API must be issued locally.

Syntax

```
#include <netcons.h>
#include <errlog.h>

NetErrorLogWrite(reserved1, code, component,
                 buf, usBuflen, insbuf,
                 nstrings, reserved2);    /* 16 bit */

Net32ErrorLogWrite(reserved1, code, component,
                  buf, ulBuflen, insbuf,
                  nstrings, reserved2);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>reserved1</i>	(unsigned char LSFAR *) must be a null pointer.
<i>code</i>	(unsigned short) specifies the error code of the network error that occurred.
<i>component</i>	(const unsigned char LSFAR *) points to an ASCIIZ string specifying which software component encountered the error.
<i>buf</i>	(const unsigned char LSFAR *) points to the raw data associated with the error condition. This parameter can be null. If non-null, this parameter points to data that is typically displayed in hexadecimal format by applications reading the error log, such as the NET.EXE application in the NET ERROR command. Examples of raw data include invalid network data or invalid configuration strings that an application encounters during initialization or execution.
<i>insbuf</i>	(const unsigned char LSFAR *) points to the ASCIIZ strings containing the error message. This parameter can be null. If non-null, this parameter points to one or more ASCIIZ strings. Applications that call either the NetErrorLogRead or Net32ErrorLogRead API use this parameter in the DosGetMessage API, which uses the ASCIIZ strings to replace any metacharacters (%1, %2, and so on) found in the message.
<i>nstrings</i>	(unsigned short) indicates the number of concatenated ASCIIZ strings that <i>insbuf</i> stores.
<i>reserved2</i>	(unsigned char LSFAR *) must be a null pointer.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.

ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_LogOverflow	2377	This log file exceeds the maximum defined size.

Remarks

The NetErrorLogWrite API internally calls the appropriate OS/2 functions to open and close the error log file.

This API issues an error log alert (with NetAlertRaise) each time an entry is written to the error log file. In addition, the NetErrorLogWrite API issues one *admin* alert when the error log file reaches 80% capacity and another when the file reaches 100% capacity. At 100% error log file capacity, NetErrorLogWrite fails, returning the error code NERR_LogOverflow.

Related Information

For information about:

- Clearing the error log file, see [Error Logging - NetErrorLogClear](#).
- Closing the error log file, see DosClose in the *OS/2 Technical Library, Programming Guide Volume 1*.
- Error codes, see [API Return Codes](#).
- Limiting the size of the error log file, see [Server - NetServerSetInfo](#).
- Retrieving the size of the error log file, see [Server - NetServerGetInfo](#).

File Category

This category includes the following APIs:

[File - NetFileClose2](#)
[File - NetFileEnum2](#)
[File - NetFileGetInfo2](#)

File APIs provide a system for monitoring which file, device, and pipe resources are opened on a server and for closing one of these resources if necessary. They are used with the SHARES.H and NETCONS.H header files.

NetFileGetInfo2 returns information about one particular opening of a resource. Two levels of detail are available: (1) the identification number assigned to the resource when it was opened (level 2), and (2) additional data on permissions, file locks, and who opened the resource (level 3).

NetFileClose2 forces a resource closed when a system error prevents normal closure by the DosClose function.

File Data Structures

The *sLevel* parameter for NetFileEnum2 and NetFileGetInfo2 specifies one of two levels of information (2 or 3) to be returned. Both APIs

return data structured as follows.

Opened Resources Level 2

```
struct file_info_2 {
  unsigned long  fi2_id;
};
```

where *fi2_id* is the identification number assigned to the resource at opening.

Opened Resources Level 3

```
struct file_info_3 {
  unsigned long      fi3_id;
  unsigned short     fi3_permissions;
  unsigned short     fi3_num_locks;
  unsigned char LSFAR * LSPTR fi3_pathname;
  unsigned char LSFAR * LSPTR fi3_username;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *fi3_permissions* indicates the access permissions of the opening application. The bit mask of *fi3_permissions* is defined in the SHARES.H header file as follows:

SYMBOLIC CONSTANT	BIT MASK	MEANING
FILE_READ	0x1	Permission to read a resource, and by default, run the resource.
FILE_WRITE	0x2	Permission to write to a resource.
FILE_CREATE	0x4	Permission to create a resource; data can be written when creating the resource.

- *fi3_num_locks* indicates the number of file locks on the file, device, or pipe.
- *fi3_pathname* points to an ASCIIZ string giving the path name of the opened resource.
- *fi3_username* points to an ASCIIZ string indicating the user that opened the resource.

NetFileClose2 or Net32FileClose2

The NetFileClose2 API forces a resource closed when a system error prevents a normal DosClose function closing.

Normally, the DosClose function is used to close a resource opened by a call to the DosOpen function. Use NetFileClose2 to force closed a resource opened by another process.

The NetFileClose2 API replaces the NetFileClose API, which is now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <shares.h>

NetFileClose2(pszServername, fileid);      /* 16 bit */

Net32FileClose2(pszServername, fileid);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

fileid (16-bit unsigned short or 32-bit unsigned long) is the identification number assigned to the resource at opening.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for

transactions.

NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_FileIdNotFound	2314	There is not an open file with that ID number.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_NoSuchServer	2460	The server ID is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about listing all open files and their identification numbers for a server, see [File - NetFileEnum2](#).

NetFileEnum2 or Net32FileEnum2

NetFileEnum2 or Net32FileEnum2

The NetFileEnum2 API supplies information about some or all open files on the server, allowing the user to supply a key to get the required information through iterated calls to the API.

The NetFileEnum2 API replaces the NetFileEnum API, which is now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <shares.h>

NetFileEnum2(pszServername, pszPath, pszUserID,
             sLevel, buf, usBuflen,
             pulEntriesReturned, pEntriesRemain,
             pResumeKey);          /* 16 bit */

Net32FileEnum2(pszServername, pszPath, pszUserID,
               ulLevel, buf, ulBuflen,
               pulEntriesReturned, pEntriesRemain,
               pResumeKey);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszPath (const unsigned char LSFAR *) points to the base path for enumeration. If the value is not NULL, the entries returned are only those whose names begin with the path to which *pszPath* points. For example, a *pszPath* pointer to C:\TMP enumerates only open files whose paths begin with C:\TMP, including C:\TMPFILE, and C:\TMP\DOCUMENT. If NULL, all open files are returned.

<i>pszUserID</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the single user whose opened files you want to enumerate. If <i>pszUserID</i> is null, the open files for all users are enumerated.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 2 or 3, specifying which data structure to use, as described in Opened Resources Level 2 and Opened Resources Level 3.
<i>pEntriesRemain</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the number of entries that have not yet been read.
<i>pResumeKey</i>	(void LSFAR *) is a pointer to the File Resume Key structure (FRK or structure <i>res_file_enum_2</i>). This field is used for continued scanning.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

This API provides a way for the user to overcome the problem that arises when the information returned exceeds 64KB. To initialize the key pResumeKey, use the macro instruction FRK_INIT supplied in the file SHARES.H, which accepts a structure FRK as an argument. The following is an example of an application code segment:

```

FRK f;

FRK_INIT( f );
NetFileEnum2 ( ..., &f);

```

When invoked with an initial *pResumeKey*, if the supplied buffer is too small to return all the requested information, the NetFileEnum2 API returns the error code ERROR_MORE_DATA and a *pResumeKey* suitable for retrieving the remaining data. When invoked with a *pResumeKey* from a previous call, it resumes the enumeration where indicated by **pResumeKey*. The user must not attempt to set this key other than to initialize it. Other values of **pResumeKey* supplied by the user must have been returned by a preceding call to NetFileEnum2.

NetFileEnum2 never returns an entry that has partial data; that is, a fixed-length data record and all variable-length data is present for each returned item. Items that cannot fit completely are not returned in the buffer. This differs from normal Enum function calls, which return partial data for some entries (usually the last few) if the buffer is too small. Enum2 is different in that the entries can be retrieved in full by subsequent calls (using the *pResumeKey*). Consequently, partial data could be misleading and is less useful than in normal Enum functions.

The *pszUserID* parameter, if not NULL, serves as a qualifier to the enumeration. The files returned are limited to those whose *pszUserID* matches the qualifier.

The *pszPath* parameter, if not NULL, serves as a prefix to qualify the enumeration. The entries returned are limited to those whose names begin with the qualifier string. For example, a *pszPath* of C:\TMP enumerates only open files whose path names begin with C:\TMP, C:\TMPFILE, and C:\TMP\DOCUMENT.

If both the *pszUserID* and the *pszPath* parameters are specified, only the files matching both the qualifying conditions are returned.

NetFileGetInfo2 or Net32FileGetInfo2

NetFileGetInfo2 or Net32FileGetInfo2

The NetFileGetInfo2 API retrieves information about a particular opening of a server resource.

The NetFileGetInfo2 API replaces the NetFileGetInfo API, which is now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```

#include <netcons.h>
#include <shares.h>

NetFileGetInfo2(pszServername, fileid, sLevel, buf,
               usBuflen, pusBytesAvail);    /* 16 bit */

Net32FileGetInfo2(pszServername, fileid, ulLevel, buf,
                 ulBuflen, pulBytesAvail);  /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>fileid</i>	(16-bit unsigned short or 32-bit unsigned long) indicates the identification number assigned to the resource at opening.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) specifies the level of detail (2 or 3) to be returned by the <i>file_info</i> data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_FileIdNotFound	2314	There is not an open file with that ID number.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Closing a file, device, or pipe, see [File - NetFileClose2](#).
- Listing files, devices, or pipes open on a server, see [File - NetFileEnum2](#).

Group Category

This category includes the following APIs:

[Group - NetGroupAdd](#)
[Group - NetGroupAddUser](#)
[Group - NetGroupDel](#)
[Group - NetGroupDelUser](#)
[Group - NetGroupEnum](#)
[Group - NetGroupGetInfo](#)
[Group - NetGroupGetUsers](#)
[Group - NetGroupSetInfo](#)
[Group - NetGroupSetUsers](#)

LAN Server and OS/2 Warp Server Only

The group APIs control user and group IDs in the LAN Server user accounts subsystem (UAS) database. They are used with the ACCESS.H and NETCONS.H header files.

A *group* is a set of users sharing common permissions in the UAS database. The group functions create or delete groups and review or adjust their membership.

Access permissions can be assigned for all members of a group by supplying the group name to the NetAccessAdd API (see [Access Permission Category](#)) instead of individually assigning each user an access permission record.

To create a user group, an application calls the NetGroupAdd API, supplying a group name. Initially, the group has no members. Members are assigned to the group by calling NetGroupAddUser.

NetGroupDelUser removes the name of a specified user from a group, and NetGroupDel disbands a group. (NetGroupDel works regardless of whether the group has members.)

Two APIs retrieve information about groups on a server. NetGroupEnum produces a list of all groups. NetGroupGetUsers lists all members of a specified group.

Directory and Security Server Only

The group APIs control user and group IDs in the Security Server in DSS. They are used with the ACCESS.H and NETCONS.H header files. Authority to execute the NetGroup APIs is based on ACLs rather than privilege level.

A *group* is a set of users in the Security Server in DSS. The group functions create or delete groups and review or adjust their membership. OTHGRPS is returned when there are one or more users in the group who are not part of the queried resource domain.

Directory and Security Server allows you to have more than 255 groups in a resource domain as long as the resource domain is not being synchronized to the NET.ACC file.

DOS Considerations

LAN Server and OS/2 Warp Server Only

A DLS peer workstation maintains its own user account database and access control lists locally. The peer workstation's user account database is never synchronized with any other user account database, such as OS/2 LAN Server domain's UAS.

Special Groups

LAN Server automatically maintains special groups that include all user IDs that are assigned any level of access authority. There are three special groups: USERS, ADMINS, and GUESTS. LAN Server places each defined user ID into one of these three groups according to the user's privilege level. Each member of these special groups must have one of the following privilege levels:

- USER_PRIV_USER
- USER_PRIV_ADMIN
- USER_PRIV_GUEST

Note: No GUEST privileges are provided in DSS.

The APIs in the group category cannot be used to delete users from these groups, nor can these groups be deleted. (Although not automatically created, this is true for members of the group LOCAL, also.) If an application calls any of the group APIs in an attempt to modify one of these special groups or one of its members, the API returns the NERR_SpeGroupOp error code.

Group Information Data Structures

This section contains the level 0 and level 1 group information data structures.

Group Information Level 0

```
struct group_info_0 {
    unsigned char      grpi0_name[GNLEN+1];
};
```

where *grpi0_name* is the name of the group.

Group Information Level 1

```
struct group_info_1 {
    unsigned char      grpi1_name[GNLEN+1];
    unsigned char      grpi1_pad_1;
    unsigned char LSFAR * LSPTR grpi1_comment;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *grpi1_pad_1* word-aligns the data structure component.

- *grpi1_comment* points to an ASCIIZ string containing the comment or remark of the group. The string can be NULL.

Group Membership Data Structure

```
struct group_users_info_0 {
    unsigned char      grui0_name[UNLEN+1];
};
```

where *grui0_name* is the name of the user in the group.

NetGroupAdd or Net32GroupAdd

NetGroupAdd or Net32GroupAdd

The NetGroupAdd API creates a new group account in the user accounts subsystem (UAS) database in LAN Server and OS/2 Warp Server.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

Groups are created in the Security Server database rather than in the NET.ACC file. User authority is based on ACLs, not privilege level.

In DSS, users and groups may have the same name.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupAdd(pszTarget, usLevel,
            buf, usBuflen);    /* 16 bit */

Net32GroupAdd(pszTarget ulLevel,
              buf, ulBuflen, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	The following describe the effects of using different forms of using <i>pszTarget</i>	
	<i>null or lservername</i>	group added to cell and server's resource

	domain
<i>./:</i> or <i>/... /cellname</i>	adds group to a cell
<i>//resdomainname</i>	group added to cell and resource domain
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which level of data structure to use, as described in Group Information Level 0 and Group Information Level 1 . Note: When a group is added at level 0, the data structure comment field is set to an empty string, since a comment field is not provided in the level 0 structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.
NERR_GroupExists	2223	The group name is already in use.

NERR_UserExists	2224	The user account already exists.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_GroupExistsInCell	7516	The group already exists in the cell.
NERR_ResDomMaxGroupEntries	7525	The resource domain already contains the maximum number of entries permitted.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Adding a user to a group, see [Group - NetGroupAddUser](#).
- Assigning group permissions, see [Access Permission - NetAccessAdd](#).
- Deleting a group account from a server, see [Group - NetGroupDel](#).
- Listing all groups on a server, see [Group - NetGroupEnum](#).

NetGroupAddUser or Net32GroupAddUser

NetGroupAddUser or Net32GroupAddUser

The NetGroupAddUser API adds a user to a group in the user accounts subsystem (UAS) database in LAN Server and OS/2 Warp Server.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

NetGroupAddUser cannot add a user to the special groups (USERS, ADMINS, or GUESTS). If an application tries to do that, this API returns the NERR_SpeGroupOp error code.

Directory and Security Server Only

User authority is based on ACLs, not privilege level.

NetGroupAddUser cannot add a user to the special groups (USERS, ADMINS, or GUESTS). If an application tries to do that, this API returns the NERR_SpeGroupOp error code.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>    /* DSS only */

NetGroupAddUser(pszTarget, pszGroupID,
                pszUserID);    /* 16 bit */

Net32GroupAddUser(pszTarget, pszGroupID,
                  pszUserID, pStatusbuf);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

pszTarget If *pszTarget* resolves to a resource domain, then both the group and the user must be defined to the resource domain. If *pszTarget* just specifies a cell, the user/group only has to be defined to the cell.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.

NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.
NERR_GroupNotFound	2220	The group does not exist.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_UserInGroup	2236	The user already belongs to this group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Related Information

For information about:

- Creating a new group, see [Group - NetGroupAdd](#).
- Defining group access permission, see [Access Permission Category](#).
- Removing a user from a group, see [Group - NetGroupDelUser](#).
- Retrieving a list of the members of a group, see [Group - NetGroupGetUsers](#).
- Setting the groups of which a user is a member, see [User - NetUserSetGroups](#).

NetGroupDel or Net32GroupDel

NetGroupDel or Net32GroupDel

The NetGroupDel API removes a group account from the user accounts subsystem (UAS) database in LAN Server and OS/2 Warp Server.

It is not necessary to remove all members from a group before deleting the group account. Deleting a group account does not delete the individual accounts of its member users.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

Groups are deleted from the Security Server Database rather than in the NET.ACC. User authority is based on ACLs, not privilege level.

If a user has *pszGroupID* specified as their primary group, the group is not deleted. An NERR_PrimaryGroup is returned under these circumstances.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupDel(pszTarget, pszGroupID);      /* 16 bit */

Net32GroupDel(pszTarget, pszGroupID, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

pszTarget If *pszTarget* resolves to a resource domain, the group must be defined to the resource domain or the NERR_GroupNotInResdom code is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_GroupNotFound	2220	The group does not exist.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

NERR_PrimaryGroup	7523	User cannot be deleted from their primary group.
-------------------	------	--

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosWrite

Related Information

For information about:

- Adding a group to the UAS database, see [Group - NetGroupAdd](#).
- Listing all groups in the UAS database, see [Group - NetGroupEnum](#).
- Removing a user from a group, see [Group - NetGroupDelUser](#).
- Retrieving a list of members for a group, see [Group - NetGroupGetUsers](#).

NetGroupDelUser or Net32GroupDelUser

NetGroupDelUser or Net32GroupDelUser

The NetGroupDelUser API removes a user from a particular group in the user accounts subsystem (UAS) database in LAN Server and OS/2 Warp Server.

Removing a user from a group does not delete the user's account in the system.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

NetGroupDelUser cannot delete a user name from the special groups (USERS, ADMINS, or GUESTS). If an application attempts to use the API to do that, NetGroupDelUser returns the NERR_SpeGroupOp error code.

Directory and Security Server Only

User authority is based on ACLs, not privilege level. If the group is the user's primary group, the API returns a NERR_PrimaryGroup error.

NetGroupDelUser cannot delete a user name from the special groups (USERS, ADMINS, or GUESTS). If an application attempts to use the API to do that, NetGroupDelUser returns the NERR_SpeGroupOp error code.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>                                /* DSS only */

NetGroupDelUser(pszTarget, pszGroupID, pszUserID);    /* 16 bit */

Net32GroupDelUser(pszTarget, pszGroupID, pszUserID,
```

```
pStatusbuf);
```

```
/* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

pszTarget If *pszTarget* resolves to a resource domain, both the group and user must be defined to the resource domain.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_GroupNotFound	2220	The group does not exist.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_UserNotInGroup	2237	The user does not belong to this

group.

NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.
NERR_PrimaryGroup	7523	User cannot be deleted from their primary group.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Related Information

For information about:

- Adding a user to a group, see [Group - NetGroupAddUser](#).
- Deleting a group, see [Group - NetGroupDel](#).
- Retrieving a list of members of a group, see [Group - NetGroupGetUsers](#).

NetGroupEnum or Net32GroupEnum

NetGroupEnum or Net32GroupEnum

The NetGroupEnum API lists all group accounts on the user accounts subsystem (UAS) database.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Only the group names can be retrieved with basic user's privilege. With administrative privilege, the comments can be returned.

Directory and Security Server Only

Any user with **r** permission on a group can retrieve both group names and comments (this ability is not restricted to administrators).

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupEnum(pszTarget, usLevel, buf, usBuflen,
             pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32GroupEnum(pszTarget, ulLevel, buf, ulBuflen,
               pulEntriesReturned, pulEntriesAvail, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which level of data structure to use, as described in Group Information Level 0 and Group Information Level 1 .
<i>pszTarget</i>	If <i>pszTarget</i> resolves to a resource domain, only the groups that are defined to the resource domain are returned. If <i>pszTarget</i> resolves to the cell, all groups in the cell (with DSS conforming names) are returned. More than 255 groups may be returned if <i>pszTarget</i> resolves to the cell or a resource domain that is not being synchronized.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been

started.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosRead

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

For DSS, allocate a large buffer and try and get everything the first time.

Related Information

For information about:

- Adding a new group to the UAS database, see [Group - NetGroupAdd](#).
- Removing a group from the UAS database, see [Group - NetGroupDel](#).

NetGroupGetInfo or Net32GroupGetInfo

NetGroupGetInfo or Net32GroupGetInfo

The NetGroupGetInfo API retrieves group-related information.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. A user without Administrative authority can call this API only with level 0 on a remote call. Users cannot issue this function on groups to which they do not belong.

Directory and Security Server Only

Users can issue this function on any group that they have **r** permission on.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupGetInfo(pszTarget, pszGroupID, usLevel,
                buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32GroupGetInfo(pszTarget, pszGroupID, ulLevel,
                  buf, ulBuflen, pulBytesAvail,
                  pStatusbuf);                    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	If <i>pszTarget</i> resolves to a resource domain, the group must be defined to the resource domain or the NERR_GroupNotInResdom error code is returned.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which level of data structure to use, as described in Group Information Level 0 and Group Information Level 1 .

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.

NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_GroupNotFound	2220	The group does not exist.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required but slow.

For DSS, allocate a large buffer and try to get everything the first time.

NetGroupGetUsers or Net32GroupGetUsers

NetGroupGetUsers or Net32GroupGetUsers

The NetGroupGetUsers API returns a list of the members to a particular group in the user accounts subsystem (UAS) database. Functionally, this API is similar to the Enum API because it enumerates the users in a group. (See [Group - NetGroupEnum](#) for related information.)

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started.

Issuing a call to NetGroupGetUsers for a special group (USERS, ADMINS, or GUESTS) requires Administrative authority. Callers with user authority can perform this function only on groups to which they belong, but users with accounts operator privilege can view the SERVERS group. User authority is allowed limited access to this API. Administrator authority is required for full access.

Directory and Security Server Only

DSS lists all members of the group (within scope). This API returns special user OTHUSERS under the following conditions:

- There are one or more users not part of the resource domain that are members of the group.
- There are one or more users with non-DSS names that are members of the group.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupGetUsers(pszTarget, pszGroupID, usLevel, buf,
                usBuflen, pusEntriesReturned, pusEntriesAvail); /* 16 bit */

Net32GroupGetUsers(pszTarget, pszGroupID, ulLevel, buf,
                  ulBuflen, pulEntriesReturned, pulEntriesAvail,
                  pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in Group Membership Data Structure.
<i>pszGroupID</i>	(const unsigned char LSFAR *) points to the ASCIIZ string specifying the name of the group whose members are returned.
<i>pszTarget</i>	If <i>pszTarget</i> resolves to a resource domain, <i>pszGroupID</i> must be defined to the resource domain. Also, only users defined to the resource domain are listed. If other users are present, the special user OTHUSERS is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.

ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_GroupNotFound	2220	The group does not exist.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosRead

Related Information

For information about listing the names of groups in the UAS database, see [Group - NetGroupEnum](#).

NetGroupSetInfo or Net32GroupSetInfo

NetGroupSetInfo or Net32GroupSetInfo

The NetGroupSetInfo API sets group-related information.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

User authority is based on ACLs, not privilege level.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupSetInfo(pszTarget, pszGroupID, usLevel,
               buf, usBuflen, parmnum);    /* 16 bit */

Net32GroupSetInfo(pszTarget, pszGroupID, ulLevel,
                 buf, ulBuflen, parmnum,
                 pStatusbuf);              /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in Group Information Level 1 .
<i>parmnum</i>	(16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a single field of the structure is to be passed. If <i>parmnum</i> is 0, the entire data structure is passed. Otherwise, <i>parmnum</i> can be set to GRP1_PARMNUM_COMMENT (numeric value of 3), where only the <i>grpi1_comment</i> field of the data structure is passed.
<i>pszTarget</i>	If <i>pszTarget</i> resolves to a resource domain, <i>pszGroupID</i> must be defined to the resource domain or NERR_GroupNotInResdom is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_GroupNotFound	2220	The group does not exist.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket

		has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg

NetGroupSetUsers or Net32GroupSetUsers

NetGroupSetUsers or Net32GroupSetUsers

The NetGroupSetUsers API sets information about users who belong to a group.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started.

Trying to set the user membership of a special group causes the NERR_SpeGroupOp error code to be returned. Administrator authority is required to call this API.

Directory and Security Server Only

User authority is based on ACLs, not privilege level. If there are non-DSS users in the group prior to this call, they are added to the group membership list automatically so they are not deleted.

Trying to set the user membership of a special group causes the NERR_SpeGroupOp error code to be returned.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetGroupSetUsers(pszTarget, pszGroupID, usLevel,
                buf, usBuflen, numentries);    /* 16 bit */

Net32GroupSetUsers(pszTarget, pszGroupID, ulLevel,
                  buf, ulBuflen, numentries,
                  pStatusbuf);                  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszGroupID</i>	(const unsigned char LSFAR *) points to the ASCIIZ string specifying the group that is to have its user information modified.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 0, specifying the data structure described in Group Membership Data Structure.

numentries (16-bit unsigned short or 32-bit unsigned long) specifies the number of data structures to be passed with this API call.

pszTarget If *pszTarget* resolves to a resource domain:

- *pszGroupID* must be defined to the resource domain.
- All users in *buf* must be members of the resource domain.
- Users in the group prior to the call that are not in the resource domain are not removed from the group by this API.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_GroupNotFound	2220	The group does not exist.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.

NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_GroupExistsInCell	7516	The group already exists in the cell.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Handle Category

This category includes the following APIs:

[Handle - NetHandleGetInfo](#)
[Handle - NetHandleSetInfo](#)

Handle APIs operate on the handles of remote serial devices and remote named pipes. They are used with the CHARDEV.H and

NETCONS.H header files.

Handle Data Structures

This section contains the data structures used by the APIs in the Handle category.

Handle Level 1

These fields are the same as the *wki0_chartime* and *wki0_charcount* fields described in [Requester Level 0](#). The values in the *wksta_info_0* data structure are used as the initial defaults for each opened handle. The NetHandle APIs allow these parameters to be inspected and tuned on a per-handle basis.

```
struct handle_info_1 {
    unsigned long    hdlil_chartime;
    unsigned short   hdlil_charcount;
};
```

where:

- *hdlil_chartime* is the amount of time (in milliseconds) the requester collects data to send to a shared serial device queue or a named pipe.
 - *hdlil_charcount* is the number of characters (in bytes) the requester stores before sending data to a serial device queue or a named pipe.
-

Handle Level 2

This level identifies the name of a user of a particular instance of a remote named pipe with multiple instances.

```
struct handle_info_2 {
    unsigned char LSFAR * LSPTR    hdlil2_username;
};
```

where *hdlil2_username* is the owner of the named pipe handle. It can be applied to a handle of the serving side of a valid remote named pipe only.

Related Information

For information about creating multiple queues for a particular serial device, see [Serial Device Category](#).

NetHandleGetInfo

NetHandleGetInfo

The NetHandleGetInfo API retrieves handle-specific information about a serial device or named pipe.

Restrictions

This API can be called from DLS and OS/2 workstations. However, DLS can issue only a level 1 call to a remote server.
This API is 16-bit only.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetHandleGetInfo(handle, sLevel, buf,
                 usBuflen, pusBytesAvail);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- handle* (unsigned short) is the handle identification number for the communication device, queue, or named pipe that is to be queried.
- sLevel* (short) can be 1 or 2, which specifies the data structure to use, as described in [Handle Data Structures](#).

Level 1 is available for handles to both serial devices and named pipes. However, level 2 is available only for named pipe handles.

When level 1 is specified, the call can be issued only from a requester. Otherwise, the ERROR_INVALID_PARAMETER error code is returned.

When level 2 is specified, the call can be issued only to a local server. At level 2, the call identifies the user of a particular instance of a remote named pipe with multiple instances. If the handle is not to a named pipe that a remote client currently has open, the ERROR_INVALID_PARAMETER error code is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length

		data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosDevIOCtl
- DosRamSemRequest
- DosFsCtl
- DosGetShrSeg
- DosQNmPipeInfo
- DosSemRequest

NetHandleSetInfo

NetHandleSetInfo

The NetHandleSetInfo API sets handle-specific information.

Restrictions

A call to this API can be issued only from a requester and only to a remote LAN Server.

If the handle is not to a remote serial device or remote named pipe, the ERROR_INVALID_PARAMETER error code is returned. An OS/2 application can control handles to remote serial devices or remote named pipes. DOS LAN Services (DLS) can control only handles to remote named pipes.

This API is 16-bit only.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetHandleSetInfo(handle, sLevel, buf,
                 usBuflen, parmnum);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>handle</i>	(unsigned short) is the handle identification number for the communication device, queue, or named pipe that is to be queried.
<i>sLevel</i>	(short) must be 1, which specifies the data structure described in Handle Level 1 .
<i>parmnum</i>	(unsigned short) specifies whether the entire data structure or only a single field in the data structure is to be passed. If the value is 0, the entire data structure is passed. As defined in the CHARDEV.H header file, <i>parmnum</i> can be set as follows to pass only a single field:

SYMBOLIC CONSTANT	VALUE	COMPONENT
HANDLE_SET_CHAR_TIME	1	hdlil_chartime

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for trans- actions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosDevIOCtl
- DosFsCtl
- DosGetShrSeg
- DosSemClear

386-HPFS Information and Administration Category

This category includes the following APIs:

- [386-HPFS and Administration - HPFS386GetInfo](#)
- [386-HPFS and Administration - NetDASDAdd](#)
- [386-HPFS and Administration - NetDASDCheck](#)
- [386-HPFS and Administration - NetDASDCtl](#)
- [386-HPFS and Administration - NetDASDDel](#)
- [386-HPFS and Administration - NetDASDEnum](#)
- [386-HPFS and Administration - NetDASDGetInfo](#)
- [386-HPFS and Administration - NetDASDSetInfo](#)

The APIs in this category control functions that are specific to the 386-HPFS file system. The HPFS386GetInfo API checks to see if 386 HPFS is installed on a particular drive, and whether Local Security is installed with it. The NetDASD APIs control directory limits functions,

which are used to limit the amount of directory space allotted to a particular user or group.

Note: Before any directory limits function can be used on a drive, that drive must be enabled (only once) for directory limits with either the NetDASDctl API, the NET DASD command, or the LAN Server Administration GUI.

See the *LAN Server Network Administrator Reference Volume 1: Planning, Installation, and Configuration* and the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks* for more information about the directory limits function, the NET DASD command, and the LAN Server Administration GUI.

DOS Considerations

Because DOS does not support HPFS, DLS requesters can issue these API calls only to remote LAN Servers.

Directory Limits Check Data Structure

This data structure is defined in the DASD.H header file and is used with the NetDASDCheck API.

```
struct dasd_check_0 {
    unsigned char LSFAR *   dc0_path;
    unsigned long            dc0_avail;
    unsigned long            dc0_usage;
};
```

where:

- *dc0_path* points to the directory that limits the space available, beginning with the drive letter.
- *dc0_avail* states the available disk space (in KB) remaining in the entire tree of the specified directory.
- *dc0_usage* states the amount of disk space (in KB) already occupied in the directory indicated by *dc0_path*.

Directory Limits Information Data Structure

This data structure is defined in the DASD.H header file and is used with these APIs:

- NetDASDAdd
- NetDASDEnum
- NetDASDGetInfo
- NetDASDSetInfo

```
struct dasd_info_0 {
    unsigned char LSFAR * LSPTR   d0_resource_name;
    unsigned long            d0_max;
    unsigned long            d0_use;
    unsigned char            d0_flag;
    unsigned char            d0_thresh;
    unsigned char            d0_delta;
};
```

where:

- *d0_resource_name* points to the name of the directory upon which the API is to act. The directory name is specified as a standard OS/2 directory path, beginning with the drive letter and ending without the backslash (\). For the NetDASDAdd and NetDASDSet APIs, the path specified by this *d0_resource_name* field can be on a redirected drive. If so, the *pszServername* parameter submitted with the NetDASDAdd or NetDASDSet call must be NULL.

- The *d0_max* field contains the amount of disk space allotted to this directory by directory limits. Specified in KB, this field can be set from 1 to 67108863. Note however, that subsequent access to this directory is functionally limited to the smaller of the following:

- Any directory limits restrictions on any parent directory
- The total free space on the drive

If the NetDASDGetInfo API returns a value of 0 to the *d0_max* field in this data structure, the specified directory (and any parent directory) does not have directory limits set.

- d0_use* is the amount (in KB) of disk space already occupied within the specified directory resource. This field is ignored for ADD or SET operations.
- The *d0_flag* field can take one of the following values:
 - DASD_VALIDATE_LIMIT_ON, which compares a newly specified limit (*d0_max* field) to the amount of disk space already taken by the directory. If the directory tree size exceeds the new limit, NERR_MaxValidationFailed is returned.
 - DASD_VALIDATE_LIMIT_OFF, which sets the new limit regardless of the present size of the directory tree.
- d0_thresh* specifies the initial alert threshold as a percentage of the total directory limit space allotted, with values ranging from 0 to 99. Zero specifies that no threshold alert is to be generated, and 99 specifies that an alert is generated when 99% of the allotted directory limit for this directory has been reached. Note that a minimum of 1KB of disk space must be allotted to a threshold; the value obtained by multiplying the *d0_thresh* decimal-point value (where 5% = 0.05) by *d0_max* must be at least 1KB.

This threshold setting generates only 1 alert when this boundary is crossed. Subsequent alerts (generated incrementally after this threshold has been reached but before the entire limit is reached) are specified according to the next field, *d0_delta*.

- d0_delta* specifies the increment in which alerts are to be generated after the *d0_thresh* threshold has been crossed but before all of the allotted disk space has been used. This field also is specified as a percentage of the total allotted directory limit space and can range from 0 upward, as long as its value is less than 99 - *d0_thresh*. Note that a minimum of 1KB of disk space must be allotted to an increment; the value obtained by multiplying the *d0_delta* decimal-point value (where 5% = 0.05) by *d0_max* must be at least 1KB.

Directory Limits Initialization Data Structure

This data structure is defined in the DASD.H header file and is used with the NetDASDCtl API.

```
struct dasd_init_0 {
    unsigned char    i0_CtlFlag;
    unsigned char    i0_Drive;
};
```

where:

- i0_CtlFlag* specifies the control function to be performed on the 386-HPFS drive. Possible values for this control flag are:

SYMBOLIC CONSTANT	VALUE	MEANING
DASD_CTL_INSTALL	1	Enables directory limits support on the specified drive.
DASD_CTL_REMOVE	2	Disables directory limits support on the specified drive.
DASD_CTL_RECALC	3	Recalculates the space used in all of the specified drive directories that have directory limits placed upon them.
- i0_Drive* specifies the letter of the drive upon which the DASD control function is to operate. This drive must reside on the server specified by the *pszServername* parameter for any API using this data structure.

386-HPFS Information Data Structure

```
struct hpfs386_info_0 {
    unsigned long    hi0_status;
};
```

where *hi0_status* is a 32-bit flag variable, mapped to the following values:

SYMBOLIC CONSTANT	BIT VALUE
HPFS386_NotRunning	0x00000000
HPFS386_Running	0x00000001
HPFS386_LocallySecure	0x00000002

NOTE: Unless otherwise noted, all fields within this 32-bit flag are independent and should, therefore, be tested using bit operations. Other values might be added in the future, so an application checking a field here should not assume that all other bits are 0.

HPFS386GetInfo16 or HPFS386GetInfo

HPFS386GetInfo16 or HPFS386GetInfo

The HPFS386GetInfo API retrieves information about whether 386 HPFS and local security are installed and running on a particular drive.

An application using this API must be linked to the dynamic link library (DLL) HPFS386.DLL. It is recommended that you link your application dynamically to this DLL. When you do this, make sure HPFS386.DLL is installed on the same workstation as your application. Otherwise, load and call this DLL during program load, thereby allowing your application to run on workstations where the DLL is not installed.

The HPFS386GetInfo API exports two additional levels of information using HPFS386.DLL. Level 1 exports drive-specific information. Level 2 exports system-wide information and is equivalent to the CACHE386 /STATS command. The header file, HPFS386.H, describes each level's structure.

Restrictions

This API can be called only from an OS/2 workstation. This API does not have any access authority requirements.

Syntax

```
#include <hpfs386.h>
#include <netcons.h>

HPFS386GetInfo16(pReserved, pszInfo, sLevel,
                buf, usBuflen, pusBytesAvail);    /* 16 bit */

HPFS386GetInfo(pReserved, pszInfo, ulLevel,
               buf, ulBuflen, pusBytesAvail);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pReserved</i>	(unsigned char LSFAR *) must be null.
<i>pszInfo</i>	(unsigned char LSFAR *) is reserved and must be null.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 0, specifying the data structure described in 386-HPFS Information Data Structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.

NetDASDAdd or Net32DASDAdd

NetDASDAdd or Net32DASDAdd

The NetDASDAdd API invokes the directory limit function, placing a limit on the amount of disk space that can be used within a directory tree. Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDCtl.](#))

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Or the caller may have **P** permission to the parent directory.

If the directory specified in the *d0_resource_name* field of the data structure already has a limit, that limit first must be removed with the NetDASDDel API.

The *d0_use* field in the data structure is ignored for this API.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDAdd(pszServername, sLevel,
           buf, usBuflen);    /* 16 bit */

Net32DASDAdd(pszServername, ulLevel,
             buf, ulBuflen);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *uLevel* (16-bit short or 32-bit unsigned long) must be 0, specifying the data structure described in [Directory Limits Information Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_DASDMaxValidationFa	12304	The supplied directory limit is smaller than the current directory size. The limit was not applied.
NERR_LimitExists	2307	The directory limit already exists.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned from the DosAllocSeg function.

Related Information

For information about:

- Enabling directory limits, see [386-HPFS and Administration - NetDASDCtl.](#)
- Changing an existing directory limit, see [386-HPFS and Administration - NetDASDSetInfo.](#)

NetDASDCheck or Net32DASDCheck

NetDASDCheck or Net32DASDCheck

The NetDASDCheck API returns the amount of disk space available and the amount already taken in a particular directory tree. Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDCtl.](#))

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Only user privilege is required to call this API.

If a redirected drive is specified for *pszDirPath*, *pszServername* must be a NULL pointer.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDCheck(pszServername, pszDirPath, sLevel,
             buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32DASDCheck(pszServername, pszDirPath, ulLevel,
               buf, ulBuflen, pulBytesAvail);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Directory Limits Check Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.

ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_LimitNotFound	2306	The directory limit was not found.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on the drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned by the DosAllocSeg function.

NetDASDCtl or Net32DASDCtl

NetDASDCtl or Net32DASDCtl

The NetDASDCtl API prepares a 386-HPFS drive for directory limits. Before any other NetDASDxxx API can be used on a drive, that drive must be enabled (only once) for directory limits by using either this API, the NET DASD command, or the LAN Server Administration GUI.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDCtl(pszServername, sLevel,
           buf, usBuflen);    /* 32 bit *slash.

Net32DASDCtl(pszServername, ulLevel,
             buf, ulBuflen);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Directory Limits Initialization Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_DASDAlreadyInstalle	2302	Directory limits are already enabled on the specified drive.
NERR_NothHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_DASDInstallVolumeLo k	2305	Support for directory limits on the specified drive cannot complete. The drive is in use or locked by another process. You must shut down and restart your system in order for directory limits to become operational.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosOpen
- DosQFSInfo

Remarks

API calls that are remoted to a server might timeout if no response is received from the server. The NetDASDCtl function might take up to several minutes to complete execution, especially on a large drive. Therefore, it can return a nonzero return code while the remote function is still running. If one of the following error codes is returned by a remote NetDASDCtl API, a timeout probably has occurred:

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_NETNAME_DELETED	64	The network name was deleted.
ERROR_BAD_NET_NAME	67	This network name cannot be found.
ERROR_VC_DISCONNECTED	240	The session was canceled.
NERR_BadTransactConfig	2141	The server is not configured for trans- actions.

These error codes do not mean necessarily that the remote NetDASDCtl function failed. The function might still be running remotely. Therefore, it is a good idea to run an application that calls NetDASDCtl on the same server where the function is to be run, so the call can be issued locally.

NetDASDDel or Net32DASDDel

NetDASDDel or Net32DASDDel

The NetDASDDel API deletes a directory limit from a specified directory. Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDCtl](#).)

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Or the caller may have **P** permission to the parent directory.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDDel(pszServername, pszDirPath);    /* 16 bit */
Net32DASDDel(pszServername, pszDirPath); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszDirPath (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the directory that is to have a directory limit deleted. (The ASCIIZ string itself is in the form of a directory path, beginning with the drive letter.)

Note: If a redirected drive is specified for *pszDirPath*, *pszServername* must be a NULL pointer.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
-------------------	-------	---------

NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_LimitNotFound	2306	The directory limit was not found.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned by the DosAllocSeg function.

NetDASDEnum or Net32DASDEnum

NetDASDEnum or Net32DASDEnum

The NetDASDEnum API returns a list of directories that have had directory limits applied to them. It returns other related information, such as the limit applied to each directory, the space used, and the alert and incremental threshold values for each directory. Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDctl.](#))

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements. However, users can only get information for the directory resources they have been granted access to.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDEnum(pszServername, pszDirPath,
            sIncludeSubdirs, sLevel, buf, usBuflen,
            pusEntriesReturned, pusEntriesAvail); /* 16 bit */

Net32DASDEnum(pszServername, pszDirPath,
              sIncludeSubdirs, ulLevel, buf, ulBuflen,
              pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszDirPath (unsigned char LSFAR *) points to an ASCIIZ string containing the directory path to be enumerated. The ASCIIZ string is in the form of a complete directory path, beginning with the drive letter, and only ending with a backslash when a root directory is specified. This API returns a data structure for each directory limit directory that begins with the specified path, which of course includes any subdirectories that have directory limits applied. If *pszDirPath* is set to NULL, this API returns all root directories and subdirectories with directory limits for each logical 386-HPFS drive.

Note: If a redirected drive is specified for *pszDirPath*, *pszServername* must be a NULL pointer.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Directory Limits Information Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The <i>sLevel</i> parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.

NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned by the DosAllocSeg function.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

NetDASDGetInfo or Net32DASDGetInfo

NetDASDGetInfo or Net32DASDGetInfo

The NetDASDGetInfo API retrieves directory limits information for a particular directory resource.

Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDctl.](#))

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Those with user authority can only get information for the directory resources to which they have been granted access. Administrator authority is required for full access to this API.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDGetInfo(pszServername, pszDirPath, sLevel,
               buf, usBuflen, pusBytesReturned);    /* 16 bit */

Net32DASDGetInfo(pszServername, pszDirPath, ulLevel,
                 buf, ulBuflen, pulBytesReturned);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszDirPath (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the directory that is to be queried. (The ASCIIZ string itself is in the form of a directory path, beginning with the drive letter.)

Note: If a redirected drive is specified for *pszDirPath*, *pszServername* must be a NULL pointer.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Directory Limits Information Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_LimitNotFound	2306	The directory limit was not found.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned by the DosAllocSeg function.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

NetDASDSetInfo or Net32DASDSetInfo

NetDASDSetInfo or Net32DASDSetInfo

The NetDASDSetInfo API sets directory limit restrictions on a specific directory resource. Before this API can be invoked, directory limits must be enabled on the 386-HPFS drive. (For details, see [386-HPFS and Administration - NetDASDCtl.](#))

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Or the caller may have **P** permission to the parent directory.

The *d0_use* field in the data structure is ignored for this API.

Syntax

```
#include <neterr.h>
#include <dasd.h>

NetDASDSetInfo(pszServername, pszDirPath, sLevel,
               buf, usBuflen);    /* 32 bit */

Net32DASDSetInfo(pszServername, pszDirPath, ulLevel,
                 buf, ulBuflen);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszDirPath (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the directory that is to be set with this call. (The ASCIIZ string itself is in the form of a complete directory path, beginning with the drive letter. This string must be identical to that specified in the *d0_resource_name* field in the data structure.)

Note: If a redirected drive is specified for *pszDirPath*, *pszServername* must be a NULL pointer.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Directory Limits Information Data Structure](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.

ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2101	The device driver is not started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_DASDMaxValidationFa	12304	The supplied directory limit is smaller than the current directory size. The limit was not applied.
NERR_LimitNotFound	2306	The directory limit was not found.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational. Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.
NERR_NetNameNotFound	2310	This shared resource does not exist.

Other codes could be returned by the DosAllocSeg function.

Mailslot Category

This category includes the following APIs:

[Mailslot - DosDeleteMailslot](#)
[Mailslot - DosMailslotInfo](#)
[Mailslot - DosMakeMailslot](#)
[Mailslot - DosPeekMailslot](#)
[Mailslot - DosReadMailslot](#)
[Mailslot - DosWriteMailslot](#)

Mailslot APIs provide one-way interprocess communication (IPC). They are used with the MAILSLOT.H and NETCONS.H header files.

Through LAN Server mailslots, data can be sent to either local or remote applications on the network. Calls to these APIs must, however, be issued locally. The Mailslot functions create and delete mailslots, retrieve information about a mailslot or a message in it, and write messages to mailslots.

Mailslots can be read or deleted only by the process that created them. Mailslots created by a process are deleted when that process ends.

An application creates a mailslot on a local computer by calling the DosMakeMailslot API and assigning the mailslot a name in the format:

`\mailslot\name`

where *name* is a unique set of characters distinguishing the mailslot from other mailslots on the computer.

The DosMakeMailslot API returns a handle to the mailslot. This handle then can be used in the following ways:

- With DosPeekMailslot to read a message in a mailslot
- With DosReadMailslot to read and remove a message
- With DosMailslotInfo to return information about a mailslot
- With DosDeleteMailslot to delete a mailslot

Any application can write messages to any mailslot on any computer on the network by calling the DosWriteMailslot API. DosWriteMailslot accepts mailslot names both in a local and remote format, as follows:

FORMAT	TYPE
\"mailslot\"\\name	Local mailslot
\\\\computername\"mailslot\"\\ name	Remote mailslot

To write data to a mailslot on a remote computer, the name of the mailslot must also include a computer name. This requirement enables multiple remote computers to use the same mailslot name locally, but to have different names on the network. (The computer name must be unique.)

Two classes of messages, *first-class* and *second-class*, can be sent to mailslots.

First-class messages can be sent to servers and peers, but not to requesters. Delivery of first-class messages is guaranteed: a message is either delivered or the sender is notified if the message is not delivered. If a mailslot is full when a first-class message arrives, DosWriteMailslot waits until DosReadMailslot reads and removes a message from the mailslot or until the delivery timeout expires (controlled by the *timeout* parameter in the DosWriteMailslot API).

Second-class messages simply are sent; no return code informs the sender of an unsuccessful delivery. This simpler delivery system tends to make second-class messages faster than first-class messages. Second-class messages sent to a destination on a local workstation are received on the workstation as first-class messages, with guaranteed delivery.

An application can write the same message to all computers on the network that have a mailslot of a particular name. Only second-class delivery is provided. By specifying an asterisk (*) for the computer name when calling DosWriteMailslot,

```
\\*\\mailslot\\name
```

sends the same message to the named mailslot on every computer in the sender's primary domain that has the locally created mailslot. There is one limitation: requesters can receive only second-class messages of up to 400 bytes in length. Servers can receive first-class or second-class messages of any size.

Messages are stored in the mailslot according to when they were received and the priority assigned them. Each message is assigned a priority from 0 (low) through 9 (high) by way of the *priority* parameter of the DosWriteMailslot API. Generally, priorities dictate the order in which messages are stored in a mailslot. High-priority messages are placed ahead of previously stored messages with the same or lower priority. However, since the OS /2 program is a multitasking operating system, this scheme cannot be guaranteed.

Mailslot messages can be read only by the process that created the mailslot.

The DosReadMailslot API reads and then removes the most current (next available) message. Since new messages might be placed in front of other messages because of priority, a process cannot guarantee that a message read by DosReadMailslot is the same message seen earlier by DosPeekMailslot.

For related information about interprocess communications (IPC) in this book, see [Named Pipe Category](#).

DosDeleteMailslot or Dos32DeleteMailslot

DosDeleteMailslot or Dos32DeleteMailslot

The DosDeleteMailslot API deletes a mailslot. This discards all messages, including any that have not been read.

Mailslots enable applications to create and store messages while a program is running. Generally, these mailslots are deleted as the last step in running a program.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally. A mailslot can be deleted only by the application that created it.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosDeleteMailslot(mailslothandle);      /* 16 bit */
Dos32DeleteMailslot(mailslothandle);    /* 32 bit */
```

Parameters

mailslothandle (unsigned long) specifies (by its handle) the mailslot to delete.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_HANDLE	6	The specified handle is not valid.

Related Information

For information about:

- Creating a mailslot, see [Mailslot - DosMakeMailslot](#).
- Obtaining information about the status of a mailslot, see [Mailslot - DosMailslotInfo](#).

DosMailslotInfo or Dos32MailslotInfo

DosMailslotInfo or Dos32MailslotInfo

The DosMailslotInfo API retrieves information about a particular mailslot.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosMailslotInfo(mailslothandle, pusMessageSize, pusMailslotSize,
               nextsize, nextpriority, msgcount);      /* 16 bit */

Dos32MailslotInfo(mailslothandle, pusMessageSize, pusMailslotSize,
                 nextsize, nextpriority, msgcount);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>mailslothandle</i>	(unsigned long) specifies (by its handle) the mailslot about which to retrieve information.
<i>pusMessageSize</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer that specifies the maximum size message (in bytes) the mailslot can accept.
<i>pusMailslotSize</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the size (in bytes) of the mailslot. The value of <i>pusMailslotSize</i> must be equal to or greater than the value of <i>pusMessageSize</i> .
<i>nextsize</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the size (in bytes) of the next message in the mailslot. If 0, no message is available.
<i>nextpriority</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the priority (0-9) of the next message in the mailslot. (This parameter is undefined if <i>nextsize</i> is 0.)
<i>msgcount</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the number of messages the mailslot contains.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_HANDLE	6	The specified handle is not valid.

Related Information

For information about:

- Creating (and obtaining the handle for) a mailslot, see [Mailslot - DosMakeMailslot](#).
- Writing a message to a mailslot, see [Mailslot - DosWriteMailslot](#).
- Retrieving the most current message in a mailslot, see [Mailslot - DosReadMailslot](#).

DosMakeMailslot or Dos32MakeMailslot

DosMakeMailslot or Dos32MakeMailslot

The DosMakeMailslot API creates a mailslot and returns its handle.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally.

Mailslot names must be unique; no two mailslots on any one computer can have the same name.

Mailslot handles cannot be passed to other processes by way of the OS/2 DosExecPgm function. Mailslot handles can be shared, however, among threads in a single process. Thus, multiple threads can use the same handle to read or write data to the mailslot.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosMakeMailslot(pMailslotName, messagesize,
               mailslotsize, pMailslotHandle);      /* 16 bit */

Dos32MakeMailslot(pMailslotName, messagesize,
                 mailslotsize, pMailslotHandle);   /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pMailslotName</i>	(const unsigned char) points to an ASCIIZ string containing the name for the mailslot. Use the format \mailslot\name.
<i>messagesize</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the maximum message size (in bytes) the mailslot can accept. Generally, mailslots cannot accept messages larger than 65475 bytes.
<i>mailslotsize</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the size (in bytes) of the mailslot. The value of <i>mailslotsize</i> must be equal to or greater than the value of <i>messagesize</i> .
<i>pMailslotHandle</i>	(unsigned long LSFAR *) points to an unsigned integer that is the returned handle for the mailslot.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.

Other codes could be returned from the following functions:

- DosAllocShrSeg
- DosExitList

DosPeekMailslot or Dos32PeekMailslot

DosPeekMailslot or Dos32PeekMailslot

The DosPeekMailslot API reads the next available message in a mailslot without removing it.

Note: If a higher-priority message arrives, there is no guarantee that a message previously read by the DosPeekMailslot API will be the same message read by a subsequent call to the DosReadMailslot API.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosPeekMailslot(mailslothandle, buf, pusBytesReturned,
               nextsize, nextpriority);    /* 16 bit */

Dos32PeekMailslot(mailslothandle, buf, pulBytesReturned,
                 nextsize, nextpriority);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>mailslothandle</i>	(unsigned long) specifies (by its handle) the mailslot to read.
<i>nextsize</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the size (in bytes) of the next message in the mailslot. If 0, no message is available.
<i>nextpriority</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the priority (0-9) of the next message in the mailslot. (This parameter is undefined if <i>nextsize</i> is 0.)

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_HANDLE	6	The specified handle is not valid.
ERROR_BROKEN_PIPE	109	Write on pipe that is not being read.

Other codes could be returned by the DosSemRequest function.

Related Information

For information about:

- Reading and removing a message, see [Mailslot - DosReadMailslot](#).
- Writing a message to a mailslot, see [Mailslot - DosWriteMailslot](#).

DosReadMailslot or Dos32ReadMailslot

DosReadMailslot or Dos32ReadMailslot

The DosReadMailslot API reads and then removes the next available message of a mailslot.

Messages are stored in a mailslot based on their priority (0-9). An incoming message with a higher priority might be stored ahead of a previously stored message with the same or lower priority.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosReadMailslot(mailslothandle, buf, pusBytesReturned,
               nextsize, nextpriority, timeout);      /* 16 bit */

Dos32ReadMailslot(mailslothandle, buf, pulBytesReturned,
                 nextsize, nextpriority, timeout);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>mailslothandle</i>	(unsigned long) specifies (by its handle) the mailslot to read.
<i>nextsize</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the size (in bytes) of the next message in the mailslot. If 0, no message is available.
<i>nextpriority</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to an unsigned short integer indicating the priority (0 through 9) of the next message in the mailslot. (This parameter is undefined if <i>nextsize</i> is 0.)
<i>timeout</i>	(long) specifies the number of milliseconds to wait if a message is not available immediately. If 0, DosReadMailslot does not wait; if -1, DosReadMailslot waits indefinitely.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_HANDL	6	The specified handle is not valid.
ERROR_INTERRUPT	95	A system call has been interrupted.
ERROR_BROKEN_PIPE	109	Write on pipe that is not being read.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.

Other codes could be returned by the DosSemRequest function.

Related Information

For information about:

- Reading a message without removing it, see [Mailslot - DosPeekMailslot](#).
- Writing a message to a mailslot, see [Mailslot - DosWriteMailslot](#).

DosWriteMailslot or Dos32WriteMailslot

DosWriteMailslot or Dos32WriteMailslot

The DosWriteMailslot API writes a message to a particular mailslot.

To send a message to all computers on the primary domain that have a local mailslot with the same name, an application must pass the *name* parameter `\\mailslot\name` and the *class* parameter 2 to DosWriteMailslot.

Restrictions

This API can be called from DLS and OS/2 workstations. A call to this API must be issued locally.

Second-class messages must be 400 bytes or smaller when written to remote requesters; they can be any size when written to local computers or remote servers.

Syntax

```
#include <netcons.h>
#include <mailslot.h>

DosWriteMailslot(pMailslotName, message, size,
                priority, class, timeout);    /* 16 bit */

Dos32WriteMailslot(pMailslotName, message, size,
                  priority, class, timeout);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pMailslotName</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the mailslot to which the message is to be written. This parameter cannot be a null pointer, but can point to a null string. For the name of a local mailslot, use the format <code>mailslot\name</code> . For a remote mailslot, use <code>\\computername\mailslot\ name</code> . For all mailslots with the same name, but on different computers in the primary domain, use <code>*mailslot\ name</code> .
<i>message</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the message to be written to the mailslot.
<i>size</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the size (in bytes) of the message to be written.
<i>priority</i>	(16-bit unsigned short or 32-bit unsigned long) assigns a priority to the message. The priority of a message ranges from 0 (lowest) to 9 (highest). High-priority messages generally are placed ahead of previously stored messages with lower priority, except when the mailslot already is full.
<i>class</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the class of mail service to be provided, where: <ul style="list-style-type: none">• First-class mail (<i>class</i> is 1) forces DosWriteMailslot to wait until a mailslot has enough room to accept <i>message</i> or until <i>timeout</i> is reached. First-class mail can be delivered only to remote servers or local computers.• Second-class mail (<i>class</i> is 2) causes DosWriteMailslot to fail if there is not enough room to write the message in the mailslot. Second-class mail can be delivered to requesters or servers.
<i>timeout</i>	(long) specifies the number of milliseconds to wait before discontinuing the attempt to write the message. A value of 0 means only one attempt is made. A value of -1 means DosWriteMailslot

repeats the attempt until the message is written (or an error occurs).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INTERRUPT	95	A system call has been interrupted.
ERROR_BROKEN_PIPE	109	Write on pipe that is not being read.
ERROR_BUFFER_OVERFLOW	111	The buffer passed to the system call is too small to hold return data.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosDevIOCtl
- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about:

- Creating a mailslot, see [Mailslot - DosMakeMailslot](#).
- Reading a message, see [Mailslot - DosReadMailslot](#).

Message Category

This category includes the following APIs:

- [Message - NetMessageBufferSend](#)
- [Message - NetMessageFileSend](#)
- [Message - NetMessageLogFileGet](#)
- [Message - NetMessageLogFileSet](#)
- [Message - NetMessageNameAdd](#)

[Message - NetMessageNameDel](#)
[Message - NetMessageNameEnum](#)
[Message - NetMessageNameFwd](#)
[Message - NetMessageNameGetInfo](#)
[Message - NetMessageNameUnFwd](#)

Message APIs are used to send, log, and forward messages. The administrator can run these APIs remotely. They are used with the MESSAGE.H and NETCONS.H header files.

A *message* is any file or buffer of data sent to a messaging name (a user or an application) on the network. To receive a message, the user or application must register its messaging name (using the NetMessageNameAdd API) in the message name table of a computer. A message name table contains a list of registered messaging names permitted to receive messages and a list of users and applications to which a message can be forwarded. Messaging names are case-sensitive and must be unique on the physical network, not just on the domain.

To delete messaging names from the message name table, use the NetMessageNameDel API.

To list all the names stored in the table, an application can call the NetMessageNameEnum API. For information about a particular user in the table, an application can call the NetMessageNameGetInfo API.

All of the message functions except NetMessageBufferSend and NetMessageFileSend require that the local computer be running the Messenger service. The NetMessageBufferSend and the NetMessageFileSend APIs only require that the *remote* computer receiving a message be running the Messenger service.

To send a message, an application can call either the NetMessageFileSend API (to send a file) or the NetMessageBufferSend API (to send a buffer of any data).

Any messages sent to a particular name also can be forwarded to another name at a different workstation through the NetMessageNameFwd API. The NetMessageNameUnFwd API ends message forwarding.

An application also can broadcast messages to all registered names on the network by specifying the *name* parameter as an asterisk (*) wildcard character for either the NetMessageFileSend API or NetMessageBufferSend API. To broadcast a message to all requesters on the LAN, have *name* point to an asterisk (*). To broadcast a message to a domain, have *name* point to a domain name followed by an asterisk (*).

Users can receive messages in one of two ways (or both at the same time):

- The received message is logged to a message log file and read later.
- The message is displayed as a pop-up message on the screen. For a pop-up message to be received, the Messenger service must be started. For more information about starting the Messenger service, see [Services Category](#).

If an application turns logging on (using NetMessageLogFileSet), all messages received for a particular user are stored in a message log file. The NetMessageLogFileGet API returns the name of a message log file of a requester or server and indicates whether message logging is enabled. The default message log file is \IBMLAN\LOGS\MESSAGES.LOG.

Each message in the log file is formatted as follows:

- An identification header containing the names of the sender and recipient and a date and time stamp
- A blank line
- The contents of the message
- A blank line
- A line containing four asterisks (****)
- A blank line

For example, here are two logged messages:

```
Message from KRISCA to AJSCHEL on Aug 04, 1990,  
14:05:20
```

```
Hello, this is a BUFFER message.
```

```
****
```

```
Message from KRISCA to AJSCHEL on Aug 04, 1990,  
14:11:48
```

```
Hello, this is a FILE message.
```

```
****
```

Note: Any process opening the message log file must open it in read-only, deny-none mode; otherwise, the Messenger service fails when

trying to log incoming messages.

DOS Considerations

In DOS, the NetMessageNameFwd and NetMessageNameUnFwd APIs only can be run remotely.

Under DOS, the *name* parameter cannot point to the name of the local requester or to the user ID that is currently logged on to that requester.

The maximum size of a message under DOS is 64KB.

Message Data Structures

The NetMessageNameEnum and NetMessageNameGetInfo APIs can accept or return data at a level 0 or level 1 of detail using the following data structures. None of the other Message APIs uses a data structure.

Message Information Level 0

```
struct msg_info_0 {
    unsigned char msgi0_name[CNLEN+1];
};
```

where *msgi0_name* is an ASCIIZ string that specifies the alias to which the message is sent.

Message Information Level 1

```
struct msg_info_1 {
    unsigned char msgi1_name[CNLEN+1];
    unsigned char msgi1_forward_flag;
    unsigned char msgi1_pad1;
    unsigned char msgi1_forward[CNLEN+1];
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *msgi1_forward_flag* specifies whether messages are sent to a name on the local computer or forwarded to a name on a remote computer. The following are the possible values:

BIT	SYMBOLIC CONSTANT	MEANING
0-1	MSGNAME_NOT_FORWARDED	Reserved; must be 0.

2	MSGNAME_FORWARDED_TO	If 1, specifies a name on a remote computer.
3	MSGNAME_NOT_FORWARDED	Reserved; must be 0.
4	MSGNAME_FORWARDED_FROM	If 1, specifies a name on the local computer.
5-7	MSGNAME_NOT_FORWARDED	Reserved; must be 0.

- *msgi1_pad1* word-aligns the data structure components.
- *msgi1_forward* is an ASCIIZ string specifying the name to which the message will be sent if messages are to be forwarded.

Related Information

For information about starting services and the Messenger service, see [Services Category](#).

NetMessageBufferSend or Net32MessageBufferSend

NetMessageBufferSend or Net32MessageBufferSend

The NetMessageBufferSend API sends a buffer of information to a registered messaging name.

Restrictions

This API can be called from DLS and OS/2 workstations. This API requires the caller to have either administrator authority or **P** permission to the parent directory. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageBufferSend(pszServername, name,
                    buf, usBuflen); /* 16 bit */

Net32MessageBufferSend(pszServername, name,
                      buf, ulBuflen); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the user or application to receive the message. To broadcast a message to all requesters on the LAN, have *name* point to an asterisk (wildcard). To broadcast a message to all users on a domain, have *name* point to a domain name followed by an asterisk (*). The *name* parameter cannot be null.

buf (unsigned char LSFAR *) points to the ASCIIZ message to be sent.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NoComputerName	2270	A computer name has not been configured.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_PausedRemote	2281	The message has been sent, but the reception is currently paused.

NERR_BadReceive	2282	The message was sent but not received.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_TruncatedBroadcast	2289	The broadcast message was truncated.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsRamSemClear
- DosFreeSeg
- DosFsCtl
- DosGetShrSeg
- DosSemClear
- DosMailslot

Remarks

For broadcast messages to the physical network, where *name* points to "" or "domain (*)", the message can be no longer than 128 bytes (and is not guaranteed to be delivered). For messages sent to all message names on a domain, the limit is 128 bytes. Otherwise, the message can be any length, provided it does not exceed the maximum receivable message size for that computer, which is set with the *sizmessbuf* parameter in the IBMLAN.INI file. The total size of *sizmessbuf* can be divided among different messages in its heap, if messages are arriving at the same time. This reduces the actual size of any single message that can be received. In addition, the *sizmessbuf* parameter can accept only limited values.

NetMessageBufferSend does not require the Messenger service to be started on a local computer.

Related Information

For information about:

- Adding a user to a message table, see [Message - NetMessageNameAdd](#).
- Using the Messenger service, see [Services Category](#).
- Sending a message file to a user, see [Message - NetMessageFileSend](#).
- Setting the *sizmessbuf* parameter of a server in the IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetMessageFileSend or Net32MessageFileSend

NetMessageFileSend or Net32MessageFileSend

The NetMessageFileSend API sends a file to a registered messaging name.

Restrictions

This API can be called from DLS and OS/2 workstations. This API requires the caller to have either administrator authority or **P** permission to the parent directory. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageFileSend(pszServername, name, pszFilename);      /* 16 bit */

Net32MessageFileSend(pszServername, name, pszFilename);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (unsigned char LSFAR *) points to the ID of the user or application to receive the file. To broadcast a file to all users on the LAN, have *name* point to an asterisk (*) wildcard. To broadcast to all users on a domain, have *name* point to a domain name followed by an asterisk (*). The *name* parameter cannot be null.

pszFilename (unsigned char LSFAR *) points to an ASCIIZ string containing the full path and the file name of the file to be sent. (For example, D:\DIRECTORY\SUBDIRECTORY\FILENAME.EXT.)

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NoComputerName	2270	A computer name has not been configured.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_PausedRemote	2281	The message has been sent, but the reception is currently paused.
NERR_BadReceive	2282	The message was sent but not received.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_TruncatedBroadcast	2289	The broadcast message was truncated.
NERR_FileError	2290	An error occurred in reading the message file.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosFsRamSemClear
- DosFreeSeg
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosRead
- DosSemClear
- DosWriteMailslot

Remarks

For broadcast messages to the physical network, where *name* points to "*" or "domain (*)", the message can be no longer than 128 bytes (and is not guaranteed to be delivered). For messages sent to all message names on a domain, the limit is 128 bytes. Otherwise, the message can be any length, provided it does not exceed the maximum receivable message size for that computer, which is set with the *sizmessbuf* parameter in the IBMLAN.INI file. The total size of *sizmessbuf* can be divided among different messages in its heap, if messages are arriving at the same time. This reduces the actual size of any single message that can be received. In addition, the *sizmessbuf* parameter can accept only limited values. For more information about the IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetMessageFileSend does not require the Messenger service to be started on a local computer.

If any special characters (for example, Ctrl+Z) are sent in a file, no information is omitted.

Related Information

For information about:

- Using the Messenger service, see [Services Category](#).

- Sending a buffer of information to a user, see [Message - NetMessageBufferSend](#).
- Setting the *sizmessbuf* parameter of a server in the IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetMessageLogFileGet or Net32MessageLogFileGet

NetMessageLogFileGet or Net32MessageLogFileGet

The NetMessageLogFileGet API retrieves the name of the message log file and the current logging status (on or off).

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. This API requires the caller to have either administrator authority or **P** permission to the parent directory. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageLogFileGet(pszServername, buf,
                    usBuflen, pSwitch);    /* 16 bit */

Net32MessageLogFileGet(pszServername, buf,
                      ulBuflen, pSwitch); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>buf</i>	(unsigned char LSFAR *) points to the name of the returned message log file. This is an ASCIIZ string containing the complete path and file name, beginning with the drive letter.
<i>pSwitch</i>	(16-bit short LSFAR * or 32-bit unsigned long LSFAR *) points to a short integer specifying whether logging is enabled. If 0, message logging is disabled; otherwise, message logging is enabled.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.

NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned by the DosFsCtl function.

Related Information

For information about:

- Using the Messenger service, see [Services Category](#).
- Changing the name and logging status of the message log file, see [Message - NetMessageLogFileSet](#).

NetMessageLogFileSet or Net32NetMessageLogFileSet

NetMessageLogFileSet or Net32NetMessageLogFileSet

The NetMessageLogFileSet API specifies the file where messages are to be logged on a particular server. This API also enables or disables the logging process.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageLogFileSet(pszServername, pFilespec, Switch); /* 16 bit */

Net32MessageLogFileSet(pszServername, pFilespec, Switch); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pFilespec (unsigned char LSFAR *) points to an ASCII string specifying the device (LPT n or COM n) or file to which the messages are logged.

If *pFilespec* is passed as a null pointer, the name of the current message log file does not change. If *pFilespec* points to a null string, no message file is used; in this case, the value of *Switch* must be 0.

If *pFilespec* points to a relative path, the path must be relative to the \BMLAN\LOGS directory. All other path names must be fully qualified. If no file name extension is provided, the LOG file extension is appended.

Switch (16-bit short or 32-bit unsigned long) specifies whether or not logging is enabled. If 0, message logging is disabled; otherwise, message logging is enabled.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_VIO_DETACHED	465	The console is not available for logging.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.

NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_InvalidDevice	2294	This is not a valid device.
NERR_WriteFault	2295	A write fault has occurred.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CantType	2357	The type of input cannot be determined.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosOpen
- DosQHandType
- DosWrite

Related Information

For information about:

- Using the Messenger service, see [Services Category](#).
- Retrieving the name and logging status of the message log file, see [Message - NetMessageLogFileGet](#).

NetMessageNameAdd or Net32MessageNameAdd

NetMessageNameAdd or Net32MessageNameAdd

The NetMessageNameAdd API registers a name in the message name table.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameAdd(pszServername, name, fwd_action); /* 16 bit */

Net32MessageNameAdd(pszServername, name, fwd_action); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (const unsigned char LSFAR *) points to an ASCIIZ string specifying a name to add to the message name table. The *name* parameter cannot be null.

fwd_action (16-bit short or 32-bit unsigned long) specifies the action to take if *name* already is forwarded. If the value is not 0, the name is added to the message name table; if the value is 0, and the name has been forwarded already, an error is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyForwarded	2274	This message alias has already been forwarded.
NERR_AddForwarded	2275	This message alias has been added, but is still forwarded.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_DeleteLater	2298	This message alias will be deleted later.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Deleting a name from a message name table, see [Message - NetMessageNameDel](#).
- Forwarding messages, see [Message - NetMessageNameFwd](#).
- Listing the names in a message name table, see [Message - NetMessageNameEnum](#).
- Using the Messenger service, see [Services Category](#).

NetMessageNameDel or Net32NetMessageNameDel

NetMessageNameDel or Net32NetMessageNameDel

The NetMessageNameDel API deletes a name from a message name table.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameDel(pszServername, name, fwd_action); /* 16 bit */
Net32MessageNameDel(pszServername, name, fwd_action); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (const unsigned char LSFAR *) points to an ASCIIZ string specifying the name to delete from the message name table. The *name* parameter cannot be null.

fwd_action (16-bit short or 32-bit unsigned long) specifies the action to take if the messages for *name* are forwarded to another name. If the value is nonzero, the forwarded name is deleted. If the value is 0, the name is not deleted.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.

NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyForwarded	2274	This message alias has already been forwarded.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_DelComputerName	2278	The computer name cannot be deleted.
NERR_NameInUse	2283	The message alias is currently in use-try again later.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_DeleteLater	2298	This message alias will be deleted later.
NERR_IncompleteDel	2299	The message alias was not successfully deleted from all networks.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Adding a name to a message name table, see [Message - NetMessageNameAdd](#).
- Listing the names in a particular message name table, see [Message - NetMessageNameEnum](#).
- Using the Messenger service, see [Services Category](#).

NetMessageNameEnum or Net32NetMessageNameEnum

NetMessageNameEnum or Net32NetMessageNameEnum

The NetMessageNameEnum API lists the name entries in a message name table.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameEnum(pszServername, sLevel, buf,
                  usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32NetMessageNameEnum(pszServername, ulLevel, buf,
                       ulBuflen, pulEntriesReturned, pulEntriesAvail);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Message Information Level 0](#) and [Message Information Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.

NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Adding a name to a message name table, see [Message - NetMessageNameAdd](#).
- Deleting a name from a message name table, see [Message - NetMessageNameDel](#).
- Using the Messenger service, see [Services Category](#).
- Retrieving information about a message account, see [Message - NetMessageNameGetInfo](#).

NetMessageNameFwd or Net32MessageNameFwd

NetMessageNameFwd or Net32MessageNameFwd

The NetMessageNameFwd API changes the message name table to forward messages to another messaging name.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameFwd(pszServername, name,
                  pszFwdName, delfor);    /* 16 bit */

Net32NetMessageNameFwd(pszServername, name,
                      pszFwdName, delfor); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>name</i>	(const unsigned char LSFAR *) points to an ASCIIZ string specifying the name receiving messages. This value cannot be null.
<i>pszFwdName</i>	(const unsigned char LSFAR *) points to an ASCIIZ string specifying the name to receive <i>name</i> 's forwarded messages. This value cannot be null.
<i>delfor</i>	(16-bit short or 32-bit unsigned long) specifies the action to take if <i>name</i> forwards messages to another name. If the value is nonzero, any previous forwarded name is deleted; if 0, any previous forwarded name is not deleted and an error is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyForwarded	2274	This message alias has already been forwarded.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_LocalForward	2279	Messages cannot be forwarded back to the same workstation.
NERR_NameInUse	2283	The message alias is currently in use-try again later.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_RemoteFull	2287	The message alias table on the remote station is full.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_DeleteLater	2298	This message alias will be deleted later.
NERR_MultipleNets	2300	This operation is not supported on machines with multiple networks.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Listing the entries in the message name table of a server, see [Message - NetMessageNameEnum](#).
- Using the Messenger service, see [Services Category](#).
- Setting the *sizmessbuf* parameter for a server in the IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.
- Stopping forwarding of a message, see [Message - NetMessageNameUnFwd](#).

NetMessageNameGetInfo or Net32MessageNameGetInfo

NetMessageNameGetInfo or Net32MessageNameGetInfo

The NetMessageNameGetInfo API retrieves information about a particular user's entry in the message name table.

Data returned at a level of 0 provides only the name of the user or application. A level 1 structure provides the name of the user and indicates whether message forwarding is available and, if so, to whom the messages are forwarded.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameGetInfo(pszServername, name, sLevel,
                     buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32MessageNameGetInfo(pszServername, name, ulLevel,
                       buf, ulBuflen, pulBytesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (const unsigned char LSFAR *) points to an ASCIIZ string specifying the name of the user of interest. This value cannot be null.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Message Information Level 0](#) and [Message Information Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.

NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about listing all names in a message name table, see [Message - NetMessageNameEnum](#).

NetMessageNameUnFwd or Net32NetMessageNameUnFwd

NetMessageNameUnFwd or Net32NetMessageNameUnFwd

The NetMessageNameUnFwd API stops forwarding of messages.

This API requires that the Messenger service be started.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <message.h>

NetMessageNameUnFwd(pszServername, name);    /* 16 bit */
Net32MessageNameUnFwd(pszServername, name);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

name (const unsigned char LSFAR *) points to an ASCIIZ string specifying the user name whose message forwarding is to be canceled. This value cannot be null.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_NameInUse	2283	The message alias is currently in use-try again later.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_RemoteFull	2287	The message alias table on the remote station is full.
NERR_NameNotForwarded	2288	Messages for this alias are not currently forwarded.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_DeleteLater	2298	This message alias will be deleted later.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Forwarding messages to another user, see [Message - NetMessageNameFwd](#).
- Listing the names in a message name table, see [Message - NetMessageNameEnum](#).
- Using the Messenger service, see [Services Category](#).

Multi-Protocol Transport Services (MPTS)

The Multi-Protocol Transport Services (MPTS) provided with LAN Server consists of two components:

- LAN Adapter and Protocol Support (LAPS)

- Socket/MPTS

The LAPS component of MPTS provides adapter, NetBIOS protocol, LAN Virtual Device Driver (VDD), and IEEE 802.2 support. LAPS contains network adapter drivers that provide communication between a protocol and network adapters, using the Network Driver Interface Specification (NDIS) as the interface. IBM and original equipment manufacturers (OEM) network adapters that comply with NDIS up to NDIS 2.01 are supported by LAPS.

The Socket/MPTS component provides a transport framework that lets Socket applications communicate using TCP/IP, NetBIOS, or Local IPC. Socket/MPTS also allows TCP/IP applications to run over NetBIOS, and NetBIOS applications to run over TCP/IP, using the non-native networking feature of Socket/MPTS.

For more information about these important functions, refer to:

- *MPTS - AnyNet for OS/2: Programmer's Reference*
- *MPTS - AnyNet for OS/2: Configuration Guide*

Named Pipe Category

This category includes the following APIs. These APIs are provided by the OS/2 base operating system and supported by LAN Server across the network. For more information about these functions, see the online information with the *IBM OS/2 Programming Toolkit* and the

FOR 16-BIT SUPPORT	FOR 32-BIT SUPPORT
DosBufReset	DosResetBuffer
DosCallNmPipe	DosCallNPipe
DosClose	
DosConnectNmPipe	DosConnectNPipe
DosDisConnectNmPipe	DosDisconnectNPipe
DosDupHandle	
DosMakeNmPipe	DosCreateNPipe
DosOpen	
DosPeekNmPipe	DosPeekNPipe
DosQFHandState	DosQueryFHState
DosQHHandType	DosQueryHType
DosQNMPHandState	DosQueryNPHState
DosQNMPipeInfo	DosQueryNPipeInfo
DosQNMPipeSemState	DosQueryNPipeSemState
DosRead	
DosReadAsync	
DosSetFHandState	DosSetFHState
DosSetNMPhandState	DosSetNPHState
DosSetNmPipeSem	DosSetNPipeSem

DosTransactNmPipe	DosTransactNPIPE
DosWaitNmPipe	DosWaitNPIPE
DosWrite	
DosWrite Async	

The following named pipe APIs, which are exclusive to DOS, are documented in [Named Pipe](#).

DosReadAsyncNmPipe
 DosWriteAsyncNmPipe

Named pipe APIs control interprocess communication (IPC) for named pipes. A *named pipe* is a bidirectional interprocess communication facility that allows two processes, either local or remote, to communicate with each other over the network.

The APIs in the named pipe category are used with the OS2.H and NETCONS.H header files. These functions are provided by the base operating system and supported by OS/2 LAN Server across the network.

A process that creates a named pipe is known as a *server process*, and a process that establishes a connection to a named pipe is known as a *client process*. To create an instance of a named pipe on the local computer, an application calls the DosMakeNmPipe API. This function specifies information that enables the server process to control the named pipe and allows client processes to access the named pipe.

To create a named pipe, DosMakeNmPipe (or 32-bit DosCreateNPIPE) requires the following:

- The name chosen for the named pipe. The format is `\pipe\name`.
- The directions (inbound, outbound, and full-duplex) in which the named pipe can send and receive data.
- An indication of whether the handle of the named pipe can be passed to spawned processes.
- The number of concurrent instances of the named pipe that can be created.
- The low-level parameters used by the OS/2 program.

An inbound or outbound named pipe (synonymous with *anonymous* pipes used with other multitasking operating systems) allows a process to read *or* write by way of one handle. A full-duplex named pipe allows a process to read *and* to write data by way of one handle.

Each time DosMakeNmPipe (or 32-bit DosCreateNPIPE) is called with the same parameter information, another instance of the named pipe is created. Each instance is associated with a unique handle, which is returned by DosMakeNmPipe (or 32-bit DosCreateNPIPE). Thus, if DosMakeNmPipe (or 32-bit DosCreateNPIPE) is called five times with the same information, five different instances (or handles) of the same named pipe are created.

You also can set other low-level operating system parameters, such as the stream type and blocking mode of the named pipe. Even though a server process creates a named pipe on a computer, client processes cannot access an instance of that named pipe until the server process calls DosConnectNmPipe (or 32-bit DosConnectNPIPE). This function informs the system that a client process has permission to access an instance of the named pipe.

To become a client process, an application opens an instance of the named pipe by calling the DosOpen API. DosOpen returns a handle to the client process that can be passed to other named pipe reading and writing functions. If DosOpen returns the ERROR_PIPE_BUSY error code (pipe currently being accessed by another process), the client process should call DosWaitNmPipe (or 32-bit DosWaitNPIPE) to wait for the named pipe to become available. DosWaitNmPipe (or 32-bit DosWaitNPIPE) can be configured either to time out after a particular period of time or to wait indefinitely for an instance of the named pipe.

When an instance of the named pipe becomes available, the DosWaitNmPipe (or 32-bit DosWaitNPIPE) API returns to the waiting client process. At this point, the client process can call DosOpen to open the named pipe. After a client process has opened an instance of a named pipe, the client process can begin to read from and write to the named pipe. To perform these tasks, the client process calls the DosRead and DosWrite APIs with the handle returned by DosOpen. If a server or client process requires that the reading and writing of named pipes be run on a separate thread, the process can call the DosReadAsync and DosWriteAsync APIs.

A remote named pipe can be written to by specifying the pipe name as `\\server\pipe\name`.

A process also can call the DosBufReset (or 32-bit DosResetBuffer) API to force all data to be written to a named pipe; normally, data written to a named pipe is held temporarily in a data buffer. The size of this buffer is specified in the initial DosMakeNmPipe (or 32-bit DosCreateNPIPE) call.

Output cannot be redirected to a named pipe.

Since a named pipe must be written to before it is read from, a process can call DosPeekNmPipe (or 32-bit DosPeekNPIPE) to see if there is any data written to a named pipe. DosPeekNmPipe (or 32-bit DosPeekNPIPE) reads the data in a named pipe but does not remove the data.

If necessary, either a server or client process can call DosDupHandle to replicate a handle to a named pipe. DosDupHandle returns a new handle to the same instance of a named pipe that an old handle represented. This handle can be passed to any named pipe function that could use the old handle.

Two APIs are provided that decrease the overhead involved in writing to and reading from a named pipe. These two APIs are DosTransactNmPipe (or 32-bit DosTransactNPIPE) and DosCallNmPipe (or 32-bit DosCallNPIPE). DosTransactNmPipe (or 32-bit DosTransactNPIPE) writes a message to and then reads a message from an opened named pipe. DosCallNmPipe (or 32-bit DosCallNPIPE) opens, writes to, reads from, and then closes a named pipe. This four-in-one process is helpful when implementing a remote-procedure call (RPC) on the network.

When a client process no longer requires access to a named pipe, the DosClose API can be called to close the named pipe.

Note: When a named pipe has been closed, DosRead and DosWrite give different return codes. DosRead returns an error code of 0 with *pcbActual* equal to 0. DosWrite returns 109 (ERROR_BROKEN_PIPE).

When a server process no longer requires an instance of a named pipe, the server process calls DosDisconnectNmPipe (or 32-bit DosDisconnectNPIPE) to remove that instance by specifying its handle. If a client process is still accessing the named pipe, DosDisconnectNmPipe (or 32-bit DosDisconnectNPIPE) forces the client process off.

The following APIs enable server or client processes to obtain information about a named pipe or its handle.

FUNCTION	PURPOSE
DosQFHandState (or 32-bit DosQueryFHState)	Determines whether the handle can be inherited and if write-behind is allowed
DosQHandType (or 32-bit DosQueryHType)	Returns the type of handle
DosQNMPHandState (or 32-bit DosQueryNPHState)	Returns the low-level parameters associated with a handle and the operating mode of the pipe; declares the instance count
DosQNMPInfo (or 32-bit DosQueryNPIPEInfo)	Returns the size of buffers and the number of instances currently available
DosQNMPSemState (or 32-bit DosQueryNPIPESemState)	Returns the state of a semaphore associated with a named pipe

The following APIs enable server or client processes to set specific information about a named pipe that can be queried.

FUNCTION	PURPOSE
DosSetFHandState (or 32-bit DosSetFHState)	Specifies whether (1) a handle of a named pipe can be inherited, and (2) write-behind is allowed
DosSetNMPHandState (or 32-bit DosSetNPHState)	Sets low-level parameters associated with pipes, such as reading and writing mode
DosSetNMPSem (or 32-bit DosSetNPIPESem)	Sets the association of a semaphore to a named pipe

The transfer mode of a named pipe is set by either DosMakeNmPipe (or 32-bit DosCreateNPIPE) or DosSetNMPHandState (or 32-bit

DosSetNPHState); a named pipe transfers data in byte-stream or message-stream mode.

A named pipe operating in byte-stream mode operates like an anonymous pipe where all data written is transferred without any special processing performed on it. When operating in message-stream mode, a named pipe can distinguish among the different messages (and size of each message) read from and written to that named pipe.

System calls operate on manual pipes and files in similar ways. Client processes use the DosOpen, DosRead, and DosClose APIs to open, read, and close both types of resources without reference to one resource being a file and the other a named pipe.

The following table describes the transition state of a named pipe, based on the action a server or client process indicates.

CURRENT STATE	ACTION	NEXT STATE
Pipe does not exist	DosMakeNmPipe (or 32-bit DosCreateNPIPE) on the server	NP_DISCONNECTED
NP_DISCONNECTED	DosConnectNmPipe (or 32-bit DosConnectNPIPE) on the server	NP_LISTENING
NP_LISTENING	DosOpen on the client	NP_CONNECTED
NP_CONNECTED	DosDisconnectNmPipe (or 32-bit DosDisconnectNPIPE) on the server	DISCONNECTED
NP_CONNECTED	DosClose on the client	NP_CLOSING
NP_CLOSING	DosDisconnectNmPipe (or 32-bit DosDisconnectNPIPE) on the server	DISCONNECTED
NP_CONNECTED	DosClose on the server	NP_CLOSING

Note: The OS/2 DosChgFilePtr API (and other APIs that perform seek operations on files) does not work with named pipes.

DOS Considerations

Under DOS, the functions can be run only on a remote server that has interprocess communication (IPC) shares. DOS applications use the NMPIPE.H header file with these APIs.

DOS supports only client processes; a pipe already must have been created and connected on a remote server. Child processes inherit the open file handles of a parent process.

Note: The Family API (FAPI) replacement library routine for DosOpen provides support for DASD opens (open Mode Flag 0x8000). Since DOS does not support this operation, pipe operations on this type of file handle return ERROR_INVALID_HANDLE rather than ERROR_BAD_PIPE.

DOS supports asynchronous reading and writing of named pipes. DosReadAsyncNmPipe does an asynchronous read from a file. It transfers the specified number of bytes from a file to a buffer asynchronously with the requesting process execution.

DosWriteAsyncNmPipe does an asynchronous write to a file. It transfers the specified number of bytes to a file from a buffer asynchronously with respect to the execution of the requesting process. (For an example of how to use DosReadAysncNmPipe and DosWriteAysncNmPipe, refer to [Notes for DOS Applications](#).)

DosBufReset (or 32-bit DosResetBuffer) returns ERROR_BROKEN_PIPE if the handle is to a named pipe that already has been closed.

Note that standard C calls to open, to close, or to perform other generic operations, can be used instead of the corresponding calls to DosOpen, close, and so on. This is also true for calls to Int 21 open, close, and other generic functions.

Related Information

For a detailed description of each function and information about anonymous pipes, named pipes, and IPC, see the *OS/2 Technical Library, Programming Guide Volume 1*.

NetBIOS Category

This category includes the following APIs:

[NetBIOS - NetBiosClose](#)
[NetBIOS - NetBiosEnum](#)
[NetBIOS - NetBiosGetInfo](#)
[NetBIOS - NetBiosOpen](#)
[NetBIOS - NetBiosSubmit](#)

The NetBIOS APIs described in this chapter are provided primarily for compatibility with existing applications. This group of APIs sometimes is referred to as the *NetBiosSubmit* interface or, simply, the *Submit* interface. These APIs are used with the NCB.H, NETBIOS.H, and NETCONS.H header files.

The LAPS component of MPTS provides an alternative NetBIOS API, which usually is the preferred API for NetBIOS programming. That API generally is referred to as the *NB30* interface to distinguish it from the *Submit* interface. For more information about the NB30 NetBIOS API, refer to the separately available *IBM LAN Technical Reference*.

Note: An application should use mailslots and pipes whenever possible, because these interfaces are less dependent on the transport implementation.

For related information in this book, see:

- [Mailslot Category](#)
- [Multi-Protocol Transport Services \(MPTS\)](#)
- [Named Pipe Category](#)

For additional information, refer to the separately available *MPTS-Configuration Guide* and *MPTS-Programmer's Reference*.

For the APIs in this chapter, applications can call the NetBiosOpen and NetBiosClose APIs to obtain and release a handle to a network device driver. This handle then is used to submit network control blocks (NCBs) to the NetBiosSubmit API to communicate between different computers on the network.

To obtain information about one or all network device drivers installed on a computer, an application can call the NetBiosGetInfo or the NetBiosEnum API, respectively.

DOS Considerations

The APIs in this category are not available to a DLS workstation.

NetBIOS Data Structures

The NetBiosOpen and NetBiosClose APIs require no data structure. NetBiosSubmit requires the standard *ncb* data structure used to define

network control blocks (NCBs).

The NetBiosEnum and NetBiosGetInfo APIs return data at either level 0 or level 1 of detail, depending on the value of the *sLevel* parameter.

NetBIOS Level 0

```
struct netbios_info_0 {
    unsigned char nb0_net_name[NETBIOS_NAME_LEN+1];
};
```

where *nb0_net_name* is an ASCIIZ string specifying the name of the network. This field is equivalent to the *netx* parameter in the IBMLAN.INI file.

NetBIOS Level 1

```
struct netbios_info_1 {
    unsigned char      nbl_net_name[NETBIOS_NAME_LEN+1];
    unsigned char      nbl_driver_name[DEVLEN+1];
    unsigned char      nbl_lana_num;
    unsigned char      nbl_pad_1;
    unsigned short     nbl_driver_type;
    unsigned short     nbl_net_status;
    unsigned long       nbl_net_bandwidth;
    unsigned short     nbl_max_sess;
    unsigned short     nbl_max_ncbs;
    unsigned short     nbl_max_names;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *nbl_driver_name* is an ASCIIZ string specifying the network device driver name the LAN Server software uses. This field is equivalent to the value of the *nb1_net_name* parameter in the Networks section of the IBMLAN.INI file.
- *nbl_lana_num* is the network adapter number used by the *nb1_driver_name*.
- *nbl_pad_1* word-aligns the data structure components.
- *nbl_driver_type* is one of two values indicating the device driver protocol type, as defined in the NETBIOS.H header file:

SYMBOLIC CONSTANT	VALUE	TYPE OF PROTOCOL
NB_TYPE_NCB	1	Network control block (NCB) protocol
NB_TYPE_MCB	2	Message control block (MCB) protocol

- *nb1_net_status* indicates the status of the device driver. The bits for this field are defined as follows:

BITS	MEANING
0	If 1, the network software is started.
1	If 1, a loopback driver is started.

2-13	Reserved.
14-15	If 0, the network software is not started. If 1, the software is operating in regular mode. If 2, the software is operating in privileged mode. If 3, the software is operating in exclusive mode.

- *nb1_net_bandwidth* indicates the number of bits per second that the network hardware accommodates.
- *nb1_max_sess* indicates the number of concurrent sessions each device driver can establish.
- *nb1_max_ncbs* indicates the number of NCBs that can be outstanding at any time.
- *nb1_max_names* indicates the number of network names a network card can establish, depending on the type of network card used.

NetBiosClose or NetBios32Close

NetBiosClose or NetBios32Close

The NetBiosClose API closes a handle to a network device driver. It ends access to the driver by invalidating its handle and canceling any outstanding network control blocks (NCBs).

Restrictions

A call to this API must be issued locally. This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <ncb.h>
#include <netbios.h>

NetBiosClose(handle, reserved);          /* 16 bit */
NetBios32Close(handle, reserved);        /* 32 bit */
```

Parameters

handle (16-bit unsigned short or 32-bit unsigned long) specifies a particular device driver.

reserved (16-bit unsigned short or 32-bit unsigned long) must be 0.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
VALIDATED_LOGON	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.

ERROR_INVALID_HANDLE	6	The specified handle is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NetBiosEnum or NetBios32Enum

NetBiosEnum or NetBios32Enum

The NetBiosEnum API returns information about all network device drivers installed on a computer.

Restrictions

This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <ncb.h>
#include <netbios.h>

NetBiosEnum(pszServername, sLevel, buf,
            usBuflen, pusEntriesReturned,
            pusEntriesAvail);          /* 16 bit */

NetBios32Enum(pszServername, ulLevel, buf,
              ulBuflen, pulEntriesReturned,
              pulEntriesAvail);      /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifies the level of detail requested for the returned data structures. (See [NetBIOS Level 0](#) and NetBIOS Level 1.)

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the

		buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

NetBiosGetInfo or NetBios32GetInfo

NetBiosGetInfo or NetBios32GetInfo

The NetBiosGetInfo API returns information about a particular network device driver installed on a computer.

Restrictions

This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <ncb.h>
#include <netbios.h>

NetBiosGetInfo(pszServername, netbiosname,
              sLevel, buf, usBuflen,
              pulBytesAvail);          /* 16 bit */

NetBios32GetInfo(pszServername, netbiosname,
                ulLevel, buf, ulBuflen,
                pulBytesAvail);      /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- netbiosname* (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the network, such as *net1*.
- sLevel*/ or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifies the level of detail requested for the returned data structures. (See [NetBIOS Level 0](#) and NetBIOS Level 1.)

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

NetBiosOpen or NetBios32Open

NetBiosOpen or NetBios32Open

The NetBiosOpen API opens a handle to a particular network device driver.

Restrictions

- A call to this API must be issued locally. This API can be called only from an OS/2 workstation.
- Before an application can open a device driver, the LAN Server redirector software must be running.
- A device driver's handle cannot be inherited by child processes. Only the process creating a device driver can pass the handle to another function.

Syntax

```
#include <netcons.h>
#include <ncb.h>
#include <netbios.h>

NetBiosOpen(devname, reserved,
           openopt, handle);    /* 16 bit */

NetBios32Open(devname, reserved,
             openopt, handle); /* 32 bit */
```

Parameters

<i>devname</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of the network, such as <i>net1</i> .
<i>reserved</i>	(unsigned char LSFAR *) is a reserved null pointer.
<i>openopt</i>	(16-bit unsigned short or 32-bit unsigned long) is defined in NETBIOS.H and specifies the access mode, which controls whether the opening process has exclusive or restricted access for the device driver, as follows:

ACCESS MODE	VALUE	MEANING
-------------	-------	---------

NB_REGULAR	1	Any number of processes can open the device driver. Does not allow reset or use of permanent NCB, nor does it receive broadcast data to-any NCBs.
NB_PRIVILEGED	2	Only one process can open the device driver. Any other processes can open the driver if they are in privileged mode. Does not allow reset or receive a permanent NCBs. NetBiosOpen fails if any other process has the device driver open in privileged or exclusive mode.
NB_EXCLUSIVE	3	Only one process can open the device driver. Any other operation is allowed. NetBiosOpen fails if any other process has an open handle to the device driver.

handle (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the device driver's returned handle.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.

NetBiosSubmit or NetBios32Submit

NetBiosSubmit or NetBios32Submit

The NetBiosSubmit API passes a network control block (NCB) packet to a network device driver.

Restrictions

A call to this API must be issued locally. This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <ncb.h>
#include <netbios.h>

NetBiosSubmit(handle, ncbopt,
             ncb);          /* 16 bit */

NetBios32Submit(handle, ncbopt,
               ncb);       /* 32 bit */
```

Parameters

<i>handle</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the handle of the network device driver. If an application needs to submit only a few NCBs and does not require a particular device driver, the application can use the reserved <i>handle</i> value 0. The 0 handle causes the API to pass an NCB to the first network device driver installed on the local computer. The first driver is processed automatically by the NetBiosOpen API in regular mode, if necessary. (Implicit calls are made to open and close the device driver.)										
<i>ncbopt</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the chaining option flags for the submitted NCB. This parameter is defined as follows: <table><tr><th>VALUE</th><th>MEANING</th></tr><tr><td>0</td><td>A single NCB is being passed.</td></tr><tr><td>1</td><td>A single NCB with error retry is being passed.</td></tr><tr><td>2</td><td>An NCB chain with proceed-on-error is being passed.</td></tr><tr><td>3</td><td>An NCB chain with stop-on-error is being passed.</td></tr></table>	VALUE	MEANING	0	A single NCB is being passed.	1	A single NCB with error retry is being passed.	2	An NCB chain with proceed-on-error is being passed.	3	An NCB chain with stop-on-error is being passed.
VALUE	MEANING										
0	A single NCB is being passed.										
1	A single NCB with error retry is being passed.										
2	An NCB chain with proceed-on-error is being passed.										
3	An NCB chain with stop-on-error is being passed.										
<i>ncb</i>	(struct ncb LSFAR *) points to an NCB data structure containing either the NCB to be run (the unchained NCB) or the link word preceding the NCB (the chained NCB). (For more information about the use of network device drivers on a LAN, refer to the <i>LAN Server Network Administrator Reference Volume 2: Performance Tuning</i> .)										

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_HANDLE	6	The specified handle is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.

Remarks

Chained NCBs must be in the same memory segment so they can be linked by a 16-bit offset pointer that precedes each NCB. An offset of 0xFFFF specifies the end of the chain.

In a proceed-on-error chain of NCBs, each NCB is run independently, regardless of any errors that might occur. Errors are returned by the last NCB in the chain.

A stop-on-error chain is ended by the LAN Server software when an NCB in the chain causes an error. The NCB's command components are set to 0xB (Command Canceled) for NCBs not processed because of an error.

For an asynchronous NCB, the value of the NCB's *ncb_post* parameter must be either 0 or the handle to a system semaphore that will be cleared upon completion of the NCB. The semaphore must not be exclusive.

The chaining option specifies whether a single NCB or an NCB chain is being passed. A single NCB can be run with optional error retry, in which case the LAN Server software reissues the NCB a set number of times in response to the following errors:

SYMBOLIC CONSTANT	VALUE	MEANING
-------------------	-------	---------

NCB_ERR_NR	9	Resource available.
NCB_ERR_SOR	12	Session open rejected.
NCB_ERR_BUSY	21	Interface busy.

Print Destination Category

This category includes the following APIs:

[Print Destination - DosPrintDestAdd](#)
[Print Destination - DosPrintDestControl](#)
[Print Destination - DosPrintDestDel](#)
[Print Destination - DosPrintDestEnum](#)
[Print Destination - DosPrintDestGetInfo](#)
[Print Destination - DosPrintDestSetInfo](#)

Print destination APIs control the printers that receive spooled print jobs on a server. A *print destination* is any spooled device, such as a line or laser printer, that physically is connected to a server. Normally, the OS/2 spooler controls print destinations. However, other applications, such as the print processor for a printer queue, also can control print destinations by calling print destination APIs.

These APIs use the NETCONS.H, NETERR.H, and PMSPL.H header files and support only 16-bit applications. For 32-bit spooler API support, see the *OS/2 Warp, Version 3 Technical Library*.

Note: For all the print APIs, DLS applications should use the DOSPRINT.H header file instead of PMSPL.H. Also, you do not need to include OS2.H or #define INCL_BASE. For example, use:

```
#include <dosprint.h>
```

for DLS applications instead of:

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
```

which is used only for 16-bit OS/2 applications.

For more information about the print APIs, see the *Presentation Manager Programming Reference Volume 1* and *Volume 2*, the *IBM OS/2 Version 1.3 Programming Guide Technical Update*, and the documentation accompanying your *IBM OS/2 Programming Toolkit*.

After a print job is submitted to a print destination, the controlling application can call DosPrintDestControl to cancel, hold, continue, or restart the current print job on the print destination.

To get information about a particular print destination, an application calls DosPrintDestGetInfo. To obtain information about all print destinations on a server, an application calls DosPrintDestEnum. You can use this API over a network to view the print destinations on a particular server.

The API print functions allow the use of more general names for print destinations. The print destination can be treated as a virtual device independent of a printer queue or a specific logical address. The DosPrintDestGetInfo and DosPrintDestEnum APIs illustrate this enhancement. Level 0 and level 1 return data only if print destinations are associated with it. The new level 2 and 3 functions return data for all print destinations. These enhanced levels are described for the print categories.

Print Destination Data Structures

This data structure has more than one level available, as follows:

Print Destination Level 0

At level 0, data is returned in the following format:

```
CHAR    szName[PDLEN+1];
```

where:

- *szName* specifies an ASCIIZ string that contains the name of a print destination.
-

Print Destination Level 1

The PRDINFO data structure has the following format:

```
typedef struct _PRDINFO {
    CHAR    szName[PDLEN+1];
    CHAR    szUserName[UNLEN+1];
    USHORT  uJobId;
    USHORT  fsStatus;
    PSZ     pszStatus;
    USHORT  time;
} PRDINFO;
```

where:

- The first field in this data structure is identical to that in the previous level.
- *szUserName* specifies an ASCIIZ string that contains the name of the print destination.
- *uJobId* specifies the identification number of the print job. The number is assigned by the print spooler.
- *fsStatus* specifies the status of the print destination.

Bits 0 and 1 have the symbolic constant PRJ_QS_QUEUED and the value 0. The bit mask isolates the print job queued status bits as follows:

BIT	SYMBOLIC CONSTANT	VALUE	MEANING
0-1	PRJ_QS_ACTIVE	0	Print job is processing.
0-1	PRJ_QS_PAUSED	1	Print job is paused.

Bits 2 through 11 indicate the print destination status. Bits 2 through 11 can be isolated using the constant PRJ_DEVSTATUS or PRD_DEVSTATUS, which has the value of 0x0FFC. Bit 15 signals whether an alert indicated that the print job was deleted. These are the meanings for the individual bits:

BIT	SYMBOLIC CONSTANT	VALUE	MEANING
2	PRJ_COMPLETE	0x0004	If 1, the print job is complete.
3	PRJ_INTERV	0x0008	If 1, intervention is required.
4	PRJ_ERROR	0x0010	If 1, an error occurred.
5	PRJ_DESTOFFLINE	0x0020	If 1, the print destination is offline.

6	PRJ_DESTPAUSED	0x0040	If 1, the print destination is paused.
7	PRJ_NOTIFY	0x0080	If 1, an alert is raised.
8	PRJ_DESTNOPAPER	0x0100	If 1, the print destination is out of paper.
9	PRJ_DESTFORMCHG	0x0200	If 1, the printer is waiting for a form change.
10	PRJ_DESTCRTCHG	0x0400	If 1, the printer is waiting for a cartridge change.
11	PRJ_DESTPENCHG	0x0800	If 1, the printer is waiting for a pen change.
15	PRJ_DELETED	0x8000	If 1, an alert indicates the print job was deleted.

- *pszStatus* points to an ASCIIZ string that contains a comment about the print destination error status.
- *time* specifies the number of minutes the current job has been printing.

Print Destination Level 2

At level 2, the *pszPrinterName* element of the PRDINFO# data structure is returned in the following format:

```
PSZ pszPrinterName;
```

where *pszPrinterName* points to an ASCIIZ string that specifies the name of a print destination. The name can have as many as CCHMAXPATHCOMP bytes, as defined in the OS/2 header file BSEDOS.H.

Print Destination Level 3

The PRDINFO3 data structure has the following format:

```
typedef struct _PRDINFO3 {
    PSZ    pszPrinterName;
    PSZ    pszUserName;
    PSZ    pszLogAddr;
    USHORT uJobId;
    USHORT fsStatus;
    PSZ    pszStatus;
    PSZ    pszComment;
    PSZ    pszDrivers;
    USHORT time;
    USHORT pad1;
} PRDINFO3;
```

where:

- The first field in this data structure is identical to that in the previous level.
- *pszUserName* points to an ASCIIZ string that specifies the name of the user who submitted the currently active job. This

variable is valid only during printing. A null string or null pointer indicates the job was submitted from the local printer.

- *pszLogAddr* points to an ASCIIZ string that specifies the name of the logical address where this printer prints, such as LPT1. If the printer is not connected to a logical address, *pszLogAddr* is a null pointer.
- The next three fields in this data structure are identical to those in the level 1 data structure.
- *pszComment* points to an ASCIIZ string that contains a print description.
- *pszDrivers* points to an ASCIIZ string that contains a list of drivers that are supported by this printer. If the device name contains blanks, the name should be enclosed in quotation marks (""). Each driver can consist of a driver name and a device name separated by a period (.). For example,

```
IBM 4029, "PSCRIPT.Apple Laserwriter"
```

indicates that this printer supports two drivers.

- *time* specifies the number of minutes the current job has been printing.
- *pad1* word-aligns the data structure component.

DosPrintDestAdd

DosPrintDestAdd

The DosPrintDestAdd API adds a print destination to the specified computer. This call creates a new printer definition.

The printer is set to print at the logical address specified by the *pszLogAddr* element of the PRDINFO3 data structure. All device drivers and queues specified with the printer already must be defined to the spooler.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

If *pszLogAddr* is a null pointer or a null string, the print destination is created but not connected to any logical address. This means the printing cannot occur on that printer or from any printer queue connected only to that printer.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintDestAdd(pszServername, uLevel, buf, usBuflen); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>uLevel</i>	(unsigned short) must be 3, which specifies the data structure described in Print Destination Level 3.
<i>buf</i>	(bytes) points to the returned data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of

return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_REQ_NOT_ACCEP	71	This request is not accepted by the network.
ERROR_INVALID_LEVEL	124	The uLevel is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DestExists	2153	The print destination already exists.
NERR_DestNoRoom	2157	No more print destinations can be added.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_DestInvalidState	2162	This operation cannot be performed on the print destination in its current state.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.
NERR_DriverNotFound	2166	The device driver specified has not been installed on the computer.
NERR_BadDev	2341	The requested device is not valid.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about changing the connections between a printer and a port, see [Print Destination - DosPrintDestSetInfo](#).

DosPrintDestControl

DosPrintDestControl

The DosPrintDestControl API pauses or continues printing on the specified print destination and cancels or restarts a print job on the specified destination.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

While paused, a print destination cannot accept any new print jobs. If the print destination is idle when the application attempts to restart or delete a print job, DosPrintDestControl returns the error code NERR_Destidle. These operations can succeed only if a job is printing.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintDestControl(pszServername, pszPrinterName, operation);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszPrinterName (const unsigned char LSFAR *) points to an ASCIIZ string specifying which print destination to control.

operation (unsigned short) specifies the operation to be performed, defined in the PMSPL.H header file as follows:

SYMBOLIC CONSTANT/BITS	VALUE	MEANING
PRD_DELETE	0	Delete the current print job.
PRD_PAUSE	1	Pause printing.
PRD_CONT	2	Continue the paused print job.
PRD_RESTART	3	Restart the print job.
	4-255	Reserved.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_DestIdle	2158	This print destination is idle and cannot accept control operations.
NERR_DestInvalidOp	2159	This print destination request contains a control function that is not valid.
NERR_ProcNoRespond	2160	The printer processor is not responding.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing the print destinations on a computer, see [Print Destination - DosPrintDestEnum](#).
- Retrieving the status of the current print job spooled to a printer, see [Print Destination - DosPrintDestGetInfo](#).

DosPrintDestDel

DosPrintDestDel

The DosPrintDestDel API deletes a print destination.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

If a job currently is printing at the print destination, DosPrintDestDel returns the error NERR_DestInvalidState.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintDestDel(pszServername, pszPrinterName);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the descriptions of the parameters that are not defined here.

pszPrinterName (const unsigned char LSFAR *) points to an ASCIIZ string specifying which print destination to delete.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_DestInvalidState	2162	This operation cannot be performed on the print destination in its current state.
NERR_InvalidComputer	2351	The specified computer name is not valid.

DosPrintDestEnum

DosPrintDestEnum

The DosPrintDestEnum API retrieves a list of all print destinations on a server, optionally supplying status information about each.

At levels 0 and 1, DosPrintDestEnum returns print destination names only if they are associated with printer queues. At levels 2 and 3, DosPrintDestEnum returns all print destinations.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```

#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintDestEnum(pszServername, uLevel, buf, usBuflen,
                 pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

```

Parameters

See [Common Parameter Definitions](#) for the descriptions of the parameters that are not defined here.

- uLevel* (unsigned short) can be 0, 1, 2, or 3, specifying which level of data structure to use, as described in [Print Destination Data Structures](#).
- pusEntriesReturned* (PUSHORT) points to an unsigned short integer specifying the number of data structures returned. This parameter is valid only if DosPrintDestEnum returns NERR_Success or ERROR_MORE_DATA.
- pusEntriesAvail* (PUSHORT) points to an unsigned short integer indicating the number of data structures that were available. This is valid only if DosPrintDestEnum returns NERR_Success or ERROR_MORE_DATA.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Pausing or continuing printing on a particular print destination, see [Print Destination - DosPrintDestControl](#).
- Retrieving the status of a particular print destination, see [Print Destination - DosPrintDestGetInfo](#).

DosPrintDestGetInfo

DosPrintDestGetInfo

The DosPrintDestGetInfo API retrieves information about a print destination.

At levels 0 and 1, DosPrintDestGetInfo returns print destination names only if they are associated with printer queues. At levels 2 and 3, DosPrintDestGetInfo returns all print destinations.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintDestGetInfo(pszServername, pszPrinterName, uLevel,
                    buf, usBuflen, pusBytesAvail); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the descriptions of the parameters that are not defined here.

pszPrinterName (const unsigned char LSFAR *) points to an ASCIIZ string specifying the print destination about which to retrieve information.

uLevel (unsigned short) specifies which level of data structure (0, 1, 2, or 3) to be returned, as described in [Print Destination Level 0](#) through Print Destination Level 3. For this API, *uLevel* works in conjunction with *pszPrinterName* and *buf* as follows:

uLevel	TYPE OF pszPrinterName	DATA RETURNED TO buf
0	Logical address	pszLogAddr field of the PRDINFO3
1	Logical address	Entire PRDINFO data structure
2	Printer name	Pointer to a printer name
3	Printer name	Entire PRDINFO3 data structure

buf (bytes) points to the returned data as described in the previous table.

pusBytesAvail (PUSHORT) points to an unsigned short integer indicating the number of bytes of information that were available. This count is valid only if DosPrintDestGetInfo returns NERR_Success, ERROR_MORE_DATA, or NERR_BufTooSmall.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing the print destinations on a computer, see [Print Destination - DosPrintDestEnum](#).
- Pausing or continuing printing on a particular print destination, see [Print Destination - DosPrintDestControl](#).

DosPrintDestSetInfo

DosPrintDestSetInfo

The DosPrintDestSetInfo API changes the configuration of a print destination.

To use this function to disconnect a printer from a port, supply a null string for the *pszLogAddr* element of the PRDINFO3 data structure.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SLENTRY
DosPrintDestSetInfo(pszServername, pszPrinterName, uLevel, buf,
                   usBuflen, parmnum); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the descriptions of the parameters that are not defined here.

pszPrinterName (const unsigned char LSFAR *) points to an ASCIIZ string specifying the print destination.

uLevel (unsigned short) specifies the PRDINFO3 data structure, described in Print Destination Level 3.

Note: The *pszPrinterName*, *uLevel*, and *buf* parameters function together for this API as they do for the DosPrintDestGetInfo API. For more information, see the description of *uLevel* in [Print Destination - DosPrintDestGetInfo](#).

buf (bytes) points to the buffer where the data structure is to be returned.

parmnum (unsigned short) determines whether *pbBuf* contains a complete PRDINFO3 data structure or a single component. If the value is 0, *uLevel* can be 1 or 3, and *pbBuf* must contain a complete PRJINFO3 data structure. Otherwise, *parmnum* must specify the ordinal position value for one of the following data structure components, defined in PMSPL.H:

SYMBOLIC CONSTANT	VALUE	COMPONENT OF PRDINFO3
PARMNUM_ALL	0	All elements
PRD_LOGADDR_PARMNUM	3	pszLogAddr
PRD_COMMENT_PARMNUM	7	pszComment
PRD_DRIVERS_PARMNUM	8	pszDrivers

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.
NERR_DriverNotFound	2166	The device driver specified has not been installed on the computer.

NERR_BadDev	2341	The requested device is not valid.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Print Job Category

This category includes the following APIs:

[Print Job - DosPrintJobContinue](#)
[Print Job - DosPrintJobDel](#)
[Print Job - DosPrintJobEnum](#)
[Print Job - DosPrintJobGetId](#)
[Print Job - DosPrintJobGetInfo](#)
[Print Job - DosPrintJobPause](#)
[Print Job - DosPrintJobSetInfo](#)

Print job APIs control the print jobs in a printer queue on a server. A *print job* is a file submitted for printing.

The print job APIs use the NETCONS.H, NETERR.H, and PMSPL.H header files, and support 16-bit applications. For 32-bit spooler API support, see the *OS/2 Warp, Version 3 Technical Library*.

Note: For all the print APIs, DLS applications should use the DOSPRINT.H header file instead of PMSPL.H. Also, you do not need to include OS2.H or #define INCL_BASE. For example, use:

```
#include <dosprint.h>
```

for DLS applications instead of:

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
```

which is used only for 16-bit OS/2 applications.

For more information about the print APIs, see the *Presentation Manager Programming Reference Volume 1* and *Volume 2*, the *IBM OS/2 Version 1.3 Programming Guide Technical Update*, and the documentation accompanying your *IBM OS/2 Programming Toolkit*.

Print jobs are identified by an integer job identifier (job). This identifier is unique on a particular computer, not only within a queue. A combination of computer name and job identifier is sufficient to identify any job on the network. A job is assigned a job identifier by the spooler when the job is queued.

Print jobs are submitted to a printer queue and stored there until a printer is available. To submit a print job to a printer queue, an application first opens the printer queue by implicitly or explicitly establishing a connection to it.

To open the same remote printer queue through an explicit connection, an application redirects a local device name (such as LPT1: or LPT2:) to the printer queue by calling NetUseAdd. The application performs an open on the redirected device name with DosOpen.

After opening the printer queue, the application can submit files to the queue by calling DosWrite. DosWrite does not accept the name of a file, but instead writes a buffer of information.

Once a print job is queued, use the print job APIs to control when the print job is printed. You can change the position of a print job in the queue, pause a print job, or delete it from the queue.

When an application no longer requires the use of a printer queue, it should close the queue by calling DosClose.

DosPrintJobSetInfo uses the level 1 data structure PRJINFO and the level 3 data structure PRJINFO3. DosPrintJobEnum uses levels 0, 1, and 2. DosPrintJobContinue uses levels 0, 1, 2, and 3.

Print Job Information Data Structures

This data structure has more than one level available, as follows:

Print Job Level 0

At level 0, data is returned in the following format:

```
USHORT    uJobId;
```

where *uJobId* specifies the identification number assigned to the print job when it was queued. This number is unique on each workstation.

Print Job Level 1

The PRJINFO data structure has the following format:

```
typedef struct _PRJINFO {
    USHORT    uJobId;
    CHAR      szUserName[UNLEN+1];
    CHAR      pad_1;
    CHAR      szNotifyName[DTLEN+1];
    CHAR      szDataType[DTLEN+1];
    PSZ       pszParms;
    USHORT    uPosition;
    USHORT    fsStatus;
    PSZ       pszStatus;
    ULONG     ulSubmitted;
    ULONG     ulSize;
    PSZ       pszComment;
} PRJINFO;
```

where:

- The first field in this data structure is identical to that in the previous level.
- *szUserName* contains an ASCIIZ string that specifies which user submitted the print job.
- *pad_1* word-aligns the data structure component.
- *szNotifyName* contains an ASCIIZ string that specifies the messaging name that receives alert messages relating to the print job.
- *szDataType* contains an ASCIIZ string that specifies the data type for the print job.
- *pszParms* points to an ASCIIZ string that contains a parameter string to pass to the spooler. The parameter string has this format:

```
parm1=value1 parm2=value2...
```

- *uPosition* specifies the position of the print job in the printer queue. If the value is 1, the print job is the next job to print.
- *fsStatus* serves as a status flag.

Bits 0 through 1 have the symbolic constant PRJ_QS_QUEUED and the value 0. The bit mask isolates the print job queued status bits as follows:

BITS	SYMBOLIC CONSTANT	VALUE	MEANING
0-1	PRJ_QS_QUEUED	0	Print job is queued.
0-1	PRJ_QS_PAUSED	1	Print job is paused.
0-1	PRJ_QS_SPOOLING	2	Print job is spooling.
0-1	PRJ_QS_PRINTING	3	Print job is printing.

Bits 2-11 indicate the print job status. Bits 2-11 can be isolated using the constant PRJ_DEVSTATUS, which has the value of 0xFFC. Bit 15 signals whether an alert indicated that the print job was deleted. These are the meanings for the individual bits:

BIT	SYMBOLIC CONSTANT	VALUE	MEANING
2	PRJ_COMPLETE	0x0004	If 1, the print job is complete.
3	PRJ_INTERV	0x0008	If 1, intervention is required.
4	PRJ_ERROR	0x0010	If 1, an error occurred.
5	PRJ_DESTOFFLINE	0x0020	If 1, the print destination is offline.
6	PRJ_DESTPAUSED	0x0040	If 1, the print destination is paused.
7	PRJ_NOTIFY	0x0080	If 1, an alert is raised.
8	PRJ_DESTNOPAPER	0x0100	If 1, the print destination is out of paper.
9	PRJ_DESTFORMCHG	0x0200	If 1, the printer is waiting for a form change.
10	PRJ_DESTCRTCHG	0x0400	If 1, the printer is waiting for a cartridge change.
11	PRJ_DESTPENCHG	0x0800	If 1, the printer is waiting for a pen change.
15	PRJ_DELETED	0x8000	If 1, an alert indicates the job was deleted.

- *pszStatus* points to an ASCII string that contains a comment about the status of the print job posted by the queue's print processor. A null pointer or null string indicates that no information was posted.
- *ulSubmitted* specifies the time the user submitted the job. The time is stored in seconds elapsed since 00:00:00, 1 January 1970.
- *ulSize* specifies the size (in bytes) of the print job.
- *pszComment* points to an ASCII string that contains a comment about the print job. The length of the comment is defined in the PMSPL.H header file.

Print Job Level 2

```
typedef struct _PRJINFO2 {
```

```

USHORT    uJobId;
USHORT    uPriority;
PSZ        pszUserName;
USHORT    uPosition;
USHORT    fsStatus;
ULONG     ulSubmitted;
ULONG     ulSize;
PSZ        pszComment;
PSZ        pszDocument;
} PRJINFO2;

```

where:

- The first field in this data structure is identical to that in the previous level.
- *uPriority* specifies the priority of the print job. The range is 1 (lowest priority) through 99 (highest priority).
- *pszUserName* points to an ASCIIZ string that specifies the name of the user who submitted the print job.
- The next 5 fields in this data structure are identical to those in the previous level.
- *pszDocument* points to an ASCIIZ string that contains the document name of the print job.

Print Job Level 3

```

typedef struct _PRJINFO3
{
    USHORT    uJobId;
    USHORT    uPriority;
    PSZ        pszUserName;
    USHORT    uPosition;
    USHORT    fsStatus;
    ULONG     ulSubmitted;
    ULONG     ulSize;
    PSZ        pszComment;
    PSZ        pszDocument;
    PSZ        pszNotifyName;
    PSZ        pszDataType;
    PSZ        pszParms;
    PSZ        pszStatus;
    PSZ        pszQueue;
    PSZ        pszQProcName;
    PSZ        pszQProcParms;
    PSZ        pszDriverName;
    PDRIVDATA pDriverData;
    PSZ        pszPrinterName;
} PRJINFO3;

```

where:

- The first 9 fields in this data structure are identical to those in the previous level.
- *pszNotifyName* points to an ASCIIZ string that contains the message alias that receives alert messages related to the print job. A null string indicates the job was submitted from the local computer.
- *pszDataType* points to an ASCIIZ string that contains the data type for the print job.
- The next 2 fields in this data structure are identical to those in Print Job Level 1.
- *pszQueue* points to an ASCIIZ string that contains the name of the printer queue that contains the print job.
- *pszQProcName* points to an ASCIIZ string that contains the name of the queue print processor.
- *pszQProcParms* points to an ASCIIZ string that contains parameters passed to the queue print processor. The parameter string format follows:

```
parm1=value1 parm2=value2...
```

- *pszDriverName* points to an ASCIIZ string that contains the name of the printer device driver.
- *pDriverData* points to the OS/2 DRIVDATA data structure for the default driver. This data is specific to the device driver and is used only if *pszDriverName* is not null. The OS/2 DRIVDATA data structure is defined in the OS2DEF.H header file.
- *pszPrinterName* points to an ASCIIZ string that contains the name of the printer on which the job is printing. If the job is not printing, *pszPrinterName* contains a null string or a null pointer.

Print Job Identification Data Structure

DosPrintJobGetId returns the PRIDINFO data structure:

```
typedef struct _PRIDINFO {
    USHORT    uJobId;
    CHAR      szServer[CNLEN+1];
    CHAR      szQName[QLEN+1];
    CHAR      pad_1;
} PRIDINFO;
```

where:

- *uJobId* specifies the identification number assigned to the print job when it was queued. The identification number is unique on a particular computer. The machine ID and *uJobId* combined are sufficient to identify a particular print job uniquely.
- *szServer* specifies an ASCIIZ string that contains the name of the machine handling the print job. The constant CNLEN is defined in the PMSPL.H header file. If the name has more than CNLEN bytes, a null string is returned.
- *szQName* specifies an ASCIIZ string that contains the name of the printer queue for the job. The constant QLEN is defined in the PMSPL.H header file. If the name has more than QLEN bytes, a null string is returned.
- *pad_1* word-aligns the data structure component.

DosPrintJobContinue

DosPrintJobContinue

The DosPrintJobContinue API continues a paused print job. A paused print job remains in the queue but is not selected to print. If it reaches the front of the queue, the spooler bypasses it and prints jobs from behind it until the job is reactivated by a DosPrintJobContinue call.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

A user with administrator privilege can release any job. Any user can release locally a job that was created locally. A user without administrator privilege can release a remote job only if the user name of the person initiating the request is the same as the user name of the person who created the job.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
```

```
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobContinue(pszServername, usJobId);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_JobNotFound	2151	The print job does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_JobInvalidState	2164	This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Obtaining the identification number of a print job, see [Print Job - DosPrintJobGetId](#).
- Pausing a print job, see [Print Job - DosPrintJobPause](#).
- Retrieving information about a particular print job, see [Print Job - DosPrintJobGetInfo](#).

DosPrintJobDel

DosPrintJobDel

The DosPrintJobDel API deletes a job from the printer queue, canceling the printing. It is possible to delete a job that currently is printing.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

A user with administrator privilege can delete any job. Any user can delete locally a job that was created locally. A user without administrator privilege can delete a remote job only if the user name of the person initiating the request is the same as the user name of the person who created the job.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobDel(pszServername, usJobId);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_JobNotFound	2151	The print job does not exist.
NERR_ProcNoRespond	2160	The printer processor is not responding.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing the print jobs in a printer queue, see [Print Job - DosPrintJobEnum](#).
- Obtaining the identification number of a print job, see [Print Job - DosPrintJobGetId](#).

DosPrintJobEnum

DosPrintJobEnum

The DosPrintJobEnum API retrieves a list of all print jobs on a specified printer queue, optionally supplying status information about each job.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Starting with LAN Server 2.0, three new print destination status bits (9, 10, and 11) were added to the *fsStatus* element of the PRJINFO data structure. Existing applications that do not examine these new status bits cannot accurately report the status.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobEnum(pszServername, pszQueueName, uLevel, buf,
                usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>uLevel</i>	(unsigned short) can be 0, 1, or 2, specifying which level of data structure to use, described in Print Job Information Data Structures .
<i>pusEntriesReturned</i>	(PUSHORT) points to an unsigned short integer indicating the number of entries that were returned. This count is valid only if DosPrintJobEnum returns NERR_Success or ERROR_MORE_DATA.
<i>pusEntriesAvail</i>	(PUSHORT) points to an unsigned short integer indicating the number of entries that were available. This count is valid only if DosPrintJobEnum returns NERR_Success or ERROR_MORE_DATA.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.

NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Changing instructions for a submitted print job, see [Print Job - DosPrintJobSetInfo](#).
- Retrieving information about a particular print job, see [Print Job - DosPrintJobGetInfo](#).

DosPrintJobGetId

DosPrintJobGetId

The DosPrintJobGetId API retrieves information about a remote print job.

Restrictions

A call to this API can be issued from DLS or OS/2 workstations, but only to the local computer. This API does not have any access authority requirements.

Although a call to this API must be issued to a local handle, that handle must be associated with a redirected device or remote queue. This API is not available for the local spooler or for other handles associated with local jobs. If this API is called from a workstation without the DLS or OS/2 requester program installed, the ERROR_NOT_SUPPORTED (50) error code is returned.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobGetId(handle, buf, usBuflen);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>handle</i>	(HFILE) contains the handle of a redirected print device.
<i>buf</i>	(unsigned char LSFAR *) points to the data structure to be used with this call, as defined in Print Job Identification Data Structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_HANDLE	6	The specified handle is not valid.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
NERR_DevNotRedirected	2107	The device is not connected.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.

Related Information

For information about:

- Listing the print jobs in a printer queue, see [Print Job - DosPrintJobEnum](#).
- Changing the instructions for a submitted print job, see [Print Job - DosPrintJobSetInfo](#).
- Retrieving information about a particular print job, see [Print Job - DosPrintJobGetInfo](#).

DosPrintJobGetInfo

DosPrintJobGetInfo

The DosPrintJobGetInfo API retrieves information about a particular print job.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobGetInfo(pszServername, usJobId, uLevel,
                  buf, usBuflen, pusBytesAvail); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

uLevel (unsigned short) can be 0, 1, 2, or 3, specifying which level of data structure to use, as described in [Print Job Information Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_JobNotFound	2151	The print job does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Listing the print jobs in a printer queue, see [Print Job - DosPrintJobEnum](#).
- Changing the instructions for a submitted print job, see [Print Job - DosPrintJobSetInfo](#).
- Obtaining the identification number of a print job, see [Print Job - DosPrintJobGetId](#).

DosPrintJobPause

DosPrintJobPause

The DosPrintJobPause API pauses a job in a printer queue.

A paused print job remains in the queue but is not selected to print. If it reaches the front of the queue, the spooler bypasses it and prints jobs from behind it until the job is reactivated by a DosPrintJobContinue call.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN

Server workstation.

A user with administrator privilege can pause any job. Any user can pause locally a job that was created locally. A user without administrator privilege can pause a remote job only if the user name of the person initiating the request is the same as the user name of the person who created the job.

DosPrintJobPause cannot pause a job that is printing. Use DosPrintDestControl instead. An application that calls DosPrintJobPause when the print job is printing returns NERR_JobInvalidState.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobPause(pszServername, usJobId);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_JobNotFound	2151	The print job does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Continuing a paused print job, see [Print Job - DosPrintJobContinue](#).
- Obtaining the identification number of a print job, see [Print Job - DosPrintJobGetId](#).
- Retrieving information about a particular print job, see [Print Job - DosPrintJobGetInfo](#).

DosPrintJobSetInfo

DosPrintJobSetInfo

The DosPrintJobSetInfo API changes the instructions for a print job. You also can use it to give a particular print job priority over other print jobs by changing either the position of the job in the printer queue or its priority.

If the spooler is restarted, the order in which jobs are put on the queue depends on the priority and age of the job. This order might be different from the order following the DosPrintJobSetInfo call.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

User authority is allowed limited access to this API. An application that does not have administrator privilege only can move jobs backward in a remote print queue or set a priority lower than the queue priority. In addition, users without administrator privilege can use DosPrintJobSetInfo only for jobs created when the same user name was logged on. A job created locally has no associated user name, and any user can set information locally for the job.

Only an administrator can set information for a job on a remote server.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintJobSetInfo(pszServername, usJobId, uLevel,
                  buf, usBuflen, parmnum); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>uLevel</i>	(unsigned short) can be 1 or 3, specifying which data structure to use, as described in Print Job Level 1 and Print Job Level 3 .
<i>parmnum</i>	(unsigned short) specifies whether the entire data structure or only a single field of the structure is to be passed. If the value is 0, the entire data structure is passed. Otherwise, <i>parmnum</i> can be set to pass only a single field of either the level 1 or level 3 data structure, defined in PMSPL.H as follows:

SYMBOLIC CONSTANT	VALUE	FIELD OF PRJINFO AND PR
PARMNUM_ALL	0	All elements
PRJ_NOTIFYNAME_PARMNUM	3	szNotifyName or pszNotif
PRJ_DATATYPE_PARMNUM	4	szDataType or pszDataTyp
PRJ_PARMS_PARMNUM	5	pszParms
PRJ_POSITION_PARMNUM	6	uPosition
PRJ_COMMENT_PARMNUM	11	pszComment

PRJ_DOCUMENT_PARMNUM	12	pszDocument (level 3 only)
PRJ_PRIORITY_PARMNUM	14	uPriority (level 3 only)
PRJ_PROCPARMS_PARMNUM	16	pszQProcParms (level 3 only)
PRJ_DRIVERDATA_PARMNUM	18	pDriverData (level 3 only)

Note: The *uPosition* field of the data structure (pointed to here with a *parmnum* value of 6) can be set to any one of the following:

VALUE	POSITION CHANGE
0	No change
1	Moves to first position
n > 1	Assumes the nth position in the queue. If the number of jobs in the queue, the job is moved to the queue.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_JobNotFound	2151	The print job does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_JobInvalidState	2164	This operation cannot be performed on the print job in its current state.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.

NERR_DriverNotFound	2166	The device driver specified has not been installed on the computer.
NERR_DriverNotFound	2168	The print processor has not been installed on the server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing the print jobs in a printer queue, see [Print Job - DosPrintJobEnum](#).
- Retrieving information about a particular print job, see [Print Job - DosPrintJobGetInfo](#).

Printer Queue Category

This category includes the following APIs:

[Printer Queue - DosPrintQAdd](#)
[Printer Queue - DosPrintQContinue](#)
[Printer Queue - DosPrintQDel](#)
[Printer Queue - DosPrintQEnum](#)
[Printer Queue - DosPrintQGetInfo](#)
[Printer Queue - DosPrintQPause](#)
[Printer Queue - DosPrintQPurge](#)
[Printer Queue - DosPrintQSetInfo](#)

Printer queue APIs control the printer queues on a server. A *printer queue* is an ordered list of submitted print jobs on a server. The DosPrintQAdd API creates a printer queue. LAN Server and the APIs in this category control printer queues.

The printer queue APIs use the NETCONS.H, NETERR.H, and PMSPL.H header files, and support 16-bit applications. For 32-bit spooler API support, see the *OS/2 Warp, Version 3 Technical Library*.

Note: For all the print APIs, DLS applications should use the DOSPRINT.H header file instead of PMSPL.H. Also, you do not need to include OS2.H or #define INCL_BASE.

For example, use:

```
#include <dosprint.h>
```

for DLS applications, instead of:

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
```

which is used only for 16-bit OS/2 applications.

For more information about the print categories, see the *Presentation Manager Programming Reference Volume 1* and *Volume 2*, the *IBM OS/2 Version 1.3 Programming Guide Technical Update*, and the documentation accompanying your *IBM OS/2 Programming Toolkit*.

A single server or workstation can have multiple printer queues.

The spooler continuously examines the printer queues, waiting for print jobs to be submitted. When a print job is submitted, the spooler determines which of several actions to take by examining certain parameters for the queue. The spooler might spool the print job directly to a print destination, or it might pass the job to a print processor for special processing before printing.

The action taken by the spooler depends on:

- Printer queue priority (in relationship to other printer queues)
- The time of day during which the printer queue accepts jobs
- The print processor specified for the printer queue
- The print destinations available to the printer queue
- Whether the print processor, driver, and driver data provide defaults for the jobs added to this queue

When a printer queue is no longer needed, an application can call `DosPrintQDel` to remove the queue. If print jobs in the printer queue remain to be processed, `DosPrintQDel` marks the printer queue as PENDING DELETE and deletes it when all jobs have been printed. If no print jobs are pending, `DosPrintQDel` immediately deletes the printer queue.

An application can pause the operation of a printer queue by calling `DosPrintQPause`. Pausing a queue suspends processing of all submitted print jobs except the current one. Print jobs can be submitted to a paused queue, but jobs are not spooled to a print destination or print processor until the printer queue is continued through a call to `DosPrintQContinue`.

To retrieve information about a particular printer queue and its state of operation, an application calls `DosPrintQGetInfo`. To obtain information about all printer queues on a server, an application calls `DosPrintQEnum`.

Under extreme circumstances, an application can cancel all pending jobs in a printer queue by calling `DosPrintQPurge`.

Printer Queue Data Structures

This section provides the data structures for the Printer Queue category.

Printer Queue Level 0

At level 0, only the name of the printer queue is returned, in the following format:

```
CHAR    szName[QNLEN+1];
```

where *szName* contains an ASCIIZ string that specifies the name of a printer queue. The constant `QNLEN` is defined in the `PMSPL.H` header file.

Printer Queue Level 1

The `PRQINFO` data structure has the following format:

```
typedef struct _PRQINFO {
    CHAR    szName[QNLEN+1];
    CHAR    pad_1;
    USHORT  uPriority;
    USHORT  uStarttime;
    USHORT  uUptime;
    PSZ     pszSepFile;
    PSZ     pszPrProc;
    PSZ     pszDestinations;
    PSZ     pszParms;
    PSZ     pszComment;
    USHORT  fsStatus;
    USHORT  cJobs;
} PRQINFO;
```

where:

- The first field in this data structure is identical to that in the previous level.
- *pad_1* word-aligns the data structure component.
- *uPriority* specifies the priority of the queue. The range is 1-9, with 1 being the highest queue priority.
- *uStarttime* is the time the queue becomes active.
- *uUntiltime* is the time the queue becomes inactive.
- *pszSepFile* is the path name of the separator page file.
- *pszPrProc* is the default queue processor name.
- *pszDestinations* is a list of print destinations for this queue.
- *pszParms* points to an ASCIIZ string that contains a parameter string to pass to the queue processor. The parameter string has the following format:

parms=value

- *pszComment* points to an ASCIIZ string that contains a comment about the queue.
- *fsStatus* specifies the status of a queue.

Bits 0 and 1 have the symbolic constant PRJ_QS_QUEUED and the value 0. The bit mask isolates the queue status bits as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
PRQ_ACTIVE	0	Active.
PRQ_PAUSE	1	Paused.
PRQ_ERROR	2	Error occurred.
PRQ_PENDING	3	Deletion pending.

- *cJobs* specifies the number of jobs currently in the queue.

Printer Queue Level 2

Level 2 is unique in that it is followed by a print job level 1 structure (described in [Print Job Level 1](#)) for every print job in the queue.

```
typedef struct _PRQINFO {
    CHAR    szName[QNLEN+1];
    CHAR    pad_1;
    USHORT  uPriority;
    USHORT  uStarttime;
    USHORT  uUntiltime;
    PSZ     pszSepFile;
    PSZ     pszPrProc;
    PSZ     pszDestinations;
    PSZ     pszParms;
    PSZ     pszComment;
    USHORT  fsStatus;
    USHORT  cJobs;
} PRQINFO;
```

where all the fields in this data structure are identical to those in the previous level.

Printer Queue Level 3

The PRQINFO3 data structure has the following format:

```
typedef struct_PRQINFO3 {
    PSZ      pszName;
    USHORT   uPriority;
    USHORT   uStarttime;
    USHORT   uUntiltime;
    USHORT   pad1;
    PSZ      pszSepFile;
    PSZ      pszPrProc;
    PSZ      pszParms;
    PSZ      pszComment;
    USHORT   fsStatus;
    USHORT   cJobs;
    PSZ      pszPrinters;
    PSZ      pszDriverName;
    PDRIVDATA pDriverData;
} PRQINFO3;
```

where:

- *pszName* points to an ASCIIZ string that contains the name of the printer queue. The length of *pszName* is defined in BSEDOS.H.
- The next 3 fields in this data structure are identical to those in the previous level.
- *pad_1* double word-aligns the data structure component.
- The next 6 fields in this data structure are identical to those in the previous level.
- *pszPrinters* points to an ASCIIZ string that contains a list of printers that can print from the printer queue.
- *pszDriverName* points to an ASCIIZ string that contains the default driver for the queue. The device driver already must have been installed.
- *pDriverData* points to the device driver data for the default driver. This data is specific to the device driver and is used only if *pszDriverName* is not a null pointer or null string.

Printer Queue Level 4

Level 4 indicates that the returned data consists of the printer queue data structure PRQINFO3, which is followed by a print job data structure PRJINFO2 for each job in the queue. The print job data structure PRJINFO2 has the following format:

```
typedef struct _PRJINFO2 {
    USHORT   uJobId;
    USHORT   uPriority;
    PSZ      pszUserName;
    USHORT   uPosition;
    USHORT   fsStatus;
    ULONG    ulSubmitted;
    ULONG    ulSize;
    PSZ      pszComment;
    PSZ      pszDocument;
} PRJINFO2;
```

where:

- *uJobId* specifies the identification number assigned to the print job when it was queued. This number is unique on each workstation.

- *pszUserName* points to an ASCIIZ string that specifies the name of the user who submitted the print job.
- *uPosition* specifies the position of the print job in the printer queue. If the value is 1, the print job is the next job to print.
- The next field in this data structure is identical to the same field in the previous level.
- *ulSubmitted* specifies the time the user submitted the job. The time is stored in seconds elapsed since 00:00:00, 1 January 1970.
- *ulSize* specifies the size (in bytes) of the print job.
- *fsStatus* serves as a status flag.

Bits 0 and 1 have the symbolic constant PRJ_QS_QUEUED and the value 0. The bit mask isolates the print job queued status bits as follows:

BITS	SYMBOLIC CONSTANT	VALUE	MEANING
0-1	PRJ_QS_QUEUED	0	Print job is queued.
0-1	PRJ_QS_PAUSED	1	Print job is paused.
0-1	PRJ_QS_SPOOLING	2	Print job is spooling.
0-1	PRJ_QS_PRINTING	3	Print job is printing.

Bits 2-11 indicate the print job status. Bits 2-11 can be isolated using the constant PRJ_DEVSTATUS, which has the value of 0xFFC. Bit 15 signals whether an alert indicated that the print job was deleted. These are the meanings for the individual bits:

BIT	SYMBOLIC CONSTANT	VALUE	MEANING
2	PRJ_COMPLETE	0x0004	If 1, the print job is complete.
3	PRJ_INTERV	0x0008	If 1, intervention is required.
4	PRJ_ERROR	0x0010	If 1, an error occurred.
5	PRJ_DESTOFFLINE	0x0020	If 1, the print destination is offline.
6	PRJ_DESTPAUSED	0x0040	If 1, the print destination is paused.
7	PRJ_NOTIFY	0x0080	If 1, an alert is raised.
8	PRJ_DESTNOPAPER	0x0100	If 1, the print destination is out of paper.
9	PRJ_DESTFORMCHG	0x0200	If 1, the printer is waiting for a form change.
10	PRJ_DESTCRTCHG	0x0400	If 1, the printer is waiting for a cartridge change.
11	PRJ_DESTPENCHG	0x0800	If 1, the printer is waiting for a pen change.
15	PRJ_DELETED	0x8000	If 1, an alert indicates the job was deleted.

- *pszDocument* points to an ASCIIZ string that contains the document name of the print job. The document name can have as many bytes as defined in the OS/2 header file, BSDOS.H.

Printer Queue Level 5

At level 5, data is returned in the following format:

PSZ pszName ;

where this field is identical to the same field in level 3.

DosPrintQAdd

DosPrintQAdd

The DosPrintQAdd API creates a printer queue on the local workstation or on a remote server. A remote server setup requires the OS/2 LAN Requester and LAN Server software.

All new applications should use the PRQINFO3 data structure by specifying the queue default *pDriverData* data value. To do this, supply a null pointer for the *pDriverData* parameter; a new *pszDriverName* element is supplied.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Creating a queue on a remote server requires administrator privilege.

If the queue name exceeds the maximum legal length, ERROR_INVALID_PARAMETER is returned. The maximum legal length depends upon the installed file system.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQAdd(pszServername, uLevel, buf, usBuflen);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of parameters not defined here.

uLevel (unsigned short) can be 1 or 3, specifying which data structure to use, as described in [Printer Queue Level 1](#) and [Printer Queue Level 3](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.

ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_QExists	2154	The printer queue already exists.
NERR_QNoRoom	2155	No more printer queues can be added.
NERR_DestNoRoom	2157	No more print destinations can be added.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_DestInvalidState	2162	This operation cannot be performed on the print destination in its current state.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.
NERR_DriverNotFound	2166	The device driver specified has not been installed on the computer.
NERR_DataTypeInvalid	2167	The data type was not supported by the queue.
NERR_DriverNotLoaded	2168	The print processor has not been installed on the server.
NERR_BadDev	2341	The requested device is not valid.
NERR_CommDevInUse	2343	This device is already in use as a communications device.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Changing the parameters of a printer queue, see [Printer Queue - DosPrintQSetInfo](#).
- Deleting a printer queue, see [Printer Queue - DosPrintQDel](#).
- Listing a server's printer queues, see [Printer Queue - DosPrintQEnum](#).
- Retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQContinue

DosPrintQContinue

The DosPrintQContinue API continues a printer queue that has been paused by a DosPrintQPause call or disabled by an error on the queue. It does not affect an active printer queue.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Continuing a queue on a remote server requires administrator privilege on the remote server.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQContinue(pszServername, pszQueueName);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_QInvalidState	2163	This operation cannot be performed on the printer queue in its current state.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing a server's printer queues, see [Printer Queue - DosPrintQEnum](#).
- Pausing a printer queue, see [Printer Queue - DosPrintQPause](#).
- Retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQDel

DosPrintQDel

The DosPrintQDel API deletes a printer queue from the spooler.

If a printer queue contains print jobs, DosPrintQDel marks the queue as PENDING DELETE so it cannot accept new print jobs. The queue is deleted when all jobs have been printed.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Deleting a queue on a remote server requires administrator privilege on the remote server.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQDel(pszServername, pszQueueName);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_QInvalidState	2163	This operation cannot be performed on the printer queue in its current state.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Canceling all print jobs in a printer queue, see [Printer Queue - DosPrintQPurge](#).
- Creating a printer queue print job, see [Printer Queue - DosPrintQAdd](#).
- Listing a server's printer queues, see [Printer Queue - DosPrintQEnum](#).
- Retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQEnum

DosPrintQEnum

The DosPrintQEnum API retrieves a list of all printer queues on a local workstation or a remote server, optionally supplying additional information.

Levels 0, 1, and 2 return queue names only if the number of bytes in the queue name is less than or equal to QNLEN, as defined in PMSPL.H. At levels 0, 1, and 2, the values of *pcReturned* and *pcTotal* are set to the count of queues with short names. At levels 3, 4, and 5, all queue names are returned, and the values *pcReturned* and *pcTotal* represent the count of all queues.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Levels 0, 1, and 2 are provided only for compatibility with existing LAN Server 1.0 applications and should not be used in new applications. All new applications should use levels 3, 4, and 5.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQEnum(pszServername, uLevel, buf, usBuflen, pusEntriesReturned,
              pusEntriesAvail);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

uLevel (unsigned short) can be 0, 1, 2, 3, 4, or 5, specifying which level of data structure to use, described under [Printer Queue Data Structures](#).

pusEntriesReturned (PUSHORT) points to an unsigned short integer indicating the number of entries that were returned. This count is valid only if NERR_Success or ERROR_MORE_DATA is returned.

pusEntriesAvail (PUSHORT) points to an unsigned short integer indicating the number of entries that were available. This count is valid only if NERR_Success or ERROR_MORE_DATA is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The uLevel is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQGetInfo

DosPrintQGetInfo

The DosPrintQGetInfo API retrieves information about a particular printer queue, and optionally, about the jobs in it.

Levels 0, 1, and 2 return queue names only if the number of bytes in the queue name is less than or equal to QNLEN, as defined in PMSPL.H. At levels 0, 1, and 2, the values of *pcReturned* and *pcTotal* are set to the count of queues with short names. At levels 3, 4, and 5, all queue names are returned, and the values *pcReturned* and *pcTotal* represent the count of all queues.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Levels 0, 1, and 2 are provided only for compatibility with existing LAN Server 1.0 applications and should not be used in new applications. All new applications should use levels 3, 4, and 5.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQGetInfo(pszServername, pszQueueName, uLevel, buf, usBuflen,
                pusBytesAvail);    /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

uLevel (unsigned short) can be 0, 1, 2, 3, 4, or 5, specifying which level of data structure to use, as described under [Printer Queue Data Structures](#).

pusEntriesAvail (PUSHORT) points to an unsigned short integer indicating the number of entries that were available. This count is valid only if NERR_Success, ERROR_MORE_DATA, or NERR_BufTooSmall is returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
ERROR_FILENAME_EXCED_RANGE	206.	The file name is longer than 8 characters or the extension is longer than

characters.

ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Determining a job's position in a printer queue, see [Print Job - DosPrintJobGetInfo](#).
- Listing a server's printer queues, see [Printer Queue - DosPrintQEnum](#).
- Changing the configuration of a printer queue, see [Printer Queue - DosPrintQSetInfo](#).

DosPrintQPause

DosPrintQPause

The DosPrintQPause API pauses a printer queue. This call suspends processing of all print jobs except for a job currently printing. Print jobs can be submitted to a paused queue, but no jobs will be printed until the queue is continued by a DosPrintQPause call.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Pausing a remote queue requires administrator privilege on the remote server.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>
```

```
SPLERR SPLENTRY
DosPrintQPause(pszServername, pszQueueName);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Continuing a paused printer queue, see [Printer Queue - DosPrintQContinue](#).
- Determining the position of a print job in a printer queue, see [Print Job - DosPrintJobGetInfo](#).
- Retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQPurge

DosPrintQPurge

The DosPrintQPurge API deletes all pending jobs in a printer queue, except any currently printing. A print job that is printing is not affected.

If a printer queue is pending deletion when DosPrintQPurge is called, the purge clears the way for the printer queue to be deleted at the end of the current printing job.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN

Server workstation. Administrator authority is required to call this API.

Purging a remote queue requires administrator privilege on the remote server.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQPurge(pszServername, pszQueueName);      /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Related Information

For information about:

- Listing a server's printer queues, see [Printer Queue - DosPrintQEnum](#).
- Listing the print jobs in a printer queue, see [Printer Queue - DosPrintQGetInfo](#).
- Retrieving information about a printer queue, see [Printer Queue - DosPrintQGetInfo](#).

DosPrintQSetInfo

DosPrintQSetInfo

The DosPrintQSetInfo API changes the configuration for a printer queue.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#define INCL_BASE
#include <os2.h>
#include <pmspl.h>
#include <neterr.h>
#include <netcons.h>

SPLERR SPLENTRY
DosPrintQSetInfo(pszServername, pszQueueName, sLevel,
                 buf, usBuflen, parmnum); /* 16 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

uLevel (unsigned short) can be 1 or 3, specifying which data structure to use, as described in [Printer Queue Level 1](#) and [Printer Queue Level 3](#).

parmnum (unsigned short) specifies whether the entire data structure or only a single field of the structure is to be passed. If the value is 0, the entire data structure is passed. Otherwise, *parmnum* can be set to pass only a single field of the level 1 or 3 data structure, defined in PMSPL.H as follows:

SYMBOLIC CONSTANT	VALUE	COMPONENT OF PRQINFO AN
PARMNUM_ALL	0	All elements
PRQ_PRIORITY_PARMNUM	3	uPriority
PRQ_STARTTIME_PARMNUM	4	uStarttime
PRQ_UNTILTIME_PARMNUM	5	uUntiltime
PRQ_SEPARATOR_PARMNUM	6	pszSepFile
PRQ_PROCESSOR_PARMNUM	11	pszPrProc
PRQ_DESTINATIONS_PARMNUM	12	pszDestinations (level 1
PRQ_PARMNS_PARMNUM	14	pszParms
PRQ_COMMENT_PARMNUM	16	pszComment
PRQ_PRINTERS_PARMNUM	18	pszPrinters (level 3 only
PRQ_DRIVERNAME_PARMNUM	18	pszDriverName (level 3 on
PRQ_DRIVERDATA_PARMNUM	18	pDriverData (level 3 only

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The uLevel parameter is not valid.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_DestNoRoom	2157	No more print destinations can be added.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_DestInvalidState	2162	This operation cannot be performed on the print destination in its current state.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.
NERR_DriverNotFound	2166	The device driver specified has not been installed on the computer.
NERR_DataTypeInvalid	2167	The data type is not supported by the processor.
NERR_DriverNotLoaded	2168	The print processor has not been installed on the server.
NERR_BadDev	2341	The requested device is not valid.
NERR_CommDevInUse	2343	This device is already in use as a communications device.

NERR_InvalidComputer

2351

The specified computer name is not valid.

Related Information

For information about:

- Creating a printer queue, see [Printer Queue - DosPrintQAdd](#).
- Listing the printer queues on a server, see [Printer Queue - DosPrintQEnum](#).

Remote Utility Category

This category includes the following APIs:

[Remote Utility - NetRemoteCopy](#)
[Remote Utility - NetRemoteExec](#)
[Remote Utility - NetRemoteMove](#)
[Remote Utility - NetRemoteTOD](#)

Remote utility APIs enable applications to copy and move remote files, remotely run a program, and access the time-of-day information on a remote server. They are used with the REMUTIL.H and NETCONS.H header files.

The NetRemoteCopy API performs optimized file copying. Files on a remote server are copied without physically moving the files to and from the local requester. The source and destination must be on the same server.

The NetRemoteMove API moves files or directories from one location to another on a remote server without physically moving the data if the source and destination are on the same drive. If the source and destination are on different drives, the move does not require shuffling the data to and from the local requester.

To run a program on a remote server, an application calls the NetRemoteExec API. NetRemoteExec performs the same tasks as the OS/2 DosExecPgm API, but on another network server.

The NetRemoteTOD API returns time-of-day information from a remote server.

DOS Considerations

Under DOS, the functions in the Remote Utility category enable applications to copy and move remote files and access the time-of-day information on a remote server.

NetRemoteCopy uses the *copy_info* data structure. NetRemoteMove uses the *move_info* data structure. NetRemoteTOD uses the *time_of_day_info* data structure. NetRemoteExec does not use a data structure.

NetRemoteCopy Data Structure

The NetRemoteCopy function returns data in the following form:

```
struct copy_info {
    unsigned short    ci_num_copied;
    unsigned char     ci_err_buf[1];
};
```

where:

- *ci_num_copied* indicates the number of files that were copied.
- *ci_err_buf* is a variable-length ASCIIZ string containing error information pertaining to the file copy.

NetRemoteMove Data Structure

The NetRemoteMove function returns data in the following form:

```
struct move_info {
    unsigned short      mi_num_moved;
    unsigned char       mi_err_buf[1];
};
```

where:

- *mi_num_moved* indicates the number of files that were moved.
- *mi_err_buf* is a variable-length ASCIIZ string containing error information pertaining to the move operation.

NetRemoteTOD Data Structure

The NetRemoteTOD function returns data in the following form:

```
struct time_of_day_info {
    unsigned long      tod_elapsedt;
    unsigned long      tod_msecs;
    unsigned char      tod_hours;
    unsigned char      tod_mins;
    unsigned char      tod_secs;
    unsigned char      tod_hunds;
    unsigned short     tod_timezone;
    unsigned short     tod_tinterval;
    unsigned char      tod_day;
    unsigned char      tod_month;
    unsigned short     tod_year;
    unsigned char      tod_weekday;
};
```

where:

- *tod_elapsedt* indicates the number of seconds that have elapsed since 1 January 1970, which is represented as 00:00:00.
- *tod_msecs* indicates the current millisecond.
- *tod_hours* indicates the current hour.
- *tod_mins* indicates the current minute.
- *tod_secs* indicates the current second.
- *tod_hunds* indicates the current hundredths of a second.
- *tod_timezone* indicates the time zone of the server, calculated (in minutes) from the Greenwich mean time (GMT) zone.
- *tod_tinterval* indicates the time interval for each tick of the clock. Each integer represents 0.0001 second.
- *tod_day* indicates the day of the month (1-31).
- *tod_month* indicates the month (1-12).

- *tod_year* indicates the year, starting with 1980.
- *tod_weekday* indicates the day of the week (0 means Sunday; 6 means Saturday).

NetRemoteCopy or Net32RemoteCopy

NetRemoteCopy or Net32RemoteCopy

The NetRemoteCopy API copies one or more files from one location to another on a remote server.

Restrictions

This API can be called from OS/2 workstations, but both the source and destination drives must be redirected drives. (See the *sourcepath* and *destpath* parameters for a description.)

Syntax

```
#include <netcons.h>
#include <remutil.h>

NetRemoteCopy(sourcepath, destpath,
               sourcepass, destpass,
               openflags, copyflags,
               buf, usBuflen);          /* 16 bit */

Net32RemoteCopy(sourcepath, destpath,
                 sourcepass, destpass,
                 openflags, copyflags,
                 buf, ulBuflen);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sourcepath (const unsigned char LSFAR *) points to an ASCIIZ string containing the path name of the files to be copied. Wildcards can be used. This parameter must begin with either a redirected drive or a UNC name.

destpath (const unsigned char LSFAR *) points to an ASCIIZ string containing the path name to which *sourcepath* is to be copied. For a wildcard *sourcepath*, *destpath* must be a directory. This parameter must begin with either a redirected drive or a UNC name.

sourcepass (const unsigned char LSFAR *) is reserved and must be NULL.

destpass (const unsigned char LSFAR *) is reserved and must be NULL.

openflags (16-bit unsigned short or 32-bit unsigned long) specifies how *destpath* will be opened. This parameter is defined as follows:

BIT	MEANING
0-1	Used if <i>destpath</i> exists. If 0, the open fails; if 1, appended; and if 2, the file is overwritten.
2-3	Reserved, with a value of 0.
4	Used if <i>destpath</i> does not exist. If 0, the open fails; file is created.

5-15 Reserved, with a value of 0.

copyflags

(16-bit unsigned short or 32-bit unsigned long) specifies how the file copy is done. This parameter is defined as follows:

BIT	MEANING
0	If 1, destpath must be a file. If bit 0 is set to 1, 0.
1	If 1, destpath must be a directory. If bit 1 is set to 1, 0.
2	If 0, destpath is opened in binary mode. If 1, destpath is opened in text mode.
3	If 0, sourcepath is opened in binary mode. If 1, sourcepath is opened in text mode.
4	If 1, all writes are verified.
5-15	Reserved.

buf

(unsigned char LSFAR *) points to the *copy_info* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NO_MORE_FILES	18	No more files are available.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_FILE_EXISTS	80	The file already exists.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NERR_SourceIsDir	2380	The source path cannot be a directory.
NERR_BadSource	2381	The source path is not valid.
NERR_BadDest	2382	The destination path is not valid.
NERR_DifferentServers	2383	The source and destination paths are on different servers.

Other codes could be returned from the following functions:

- DosFsCtl
- DosDevIOCtl

Remarks

Currently, the source and destination for the file copy must be on the same server, or an error results. The following cases are valid:

- The source and destination are both files. The source file is copied to the destination file, subject to *openflags* and *copyflags* limitations.
- The source is either a file or wildcard, and the destination is a directory. The source files are copied to the destination directory, subject to *openflags* and *copyflags* limitations.

Related Information

For information about:

- Listing the shared resources of a server, see [Share - NetShareEnum](#).
- Moving remote files between servers, see [Remote Utility - NetRemoteMove](#).
- Share passwords, see [Share Category](#).

NetRemoteExec or Net32RemoteExec

NetRemoteExec or Net32RemoteExec

The NetRemoteExec API runs a program located on a remote server.

Restrictions

This API can be called only from an OS/2 workstation, and *objnamebuf* must point to a redirected resource.

Syntax

```
#include <netcons.h>
#include <remutil.h>

NetRemoteExec(reserved1, objnamebuf, objnamebufl,
              asyntraceflags, pArg, envpointer,
              returncodes, pgmPointer, reserved2,
              remexecflags); /* 16 bit */

Net32RemoteExec(reserved1, objnamebuf, objnamebufl,
                asyntraceflags, pArg, envpointer,
                returncodes, pgmPointer, reserved2,
                remexecflags); /* 32 bit */
```

Parameters

<i>reserved1</i>	(const unsigned char LSFAR *) is a reserved pointer with the value of -1.								
<i>objnamebuf</i>	(unsigned char LSFAR *) points to the name of the object, such as a dynamic link library. The NetRemoteExec function copies a name to this buffer if it could not successfully load and start the specified program.								
<i>objnamebuf1</i>	(unsigned LSINT) specifies the size (in bytes) of the <i>objnamebuf</i> memory area.								
<i>asyntraceflags</i>	(unsigned LSINT) specifies the asynchronous and trace flags. This parameter is defined as follows: <table><tr><td>VALUE</td><td>MEANING</td></tr><tr><td>0</td><td>Synchronous process</td></tr><tr><td>1</td><td>Asynchronous process without result code</td></tr><tr><td>2</td><td>Asynchronous process with result code</td></tr></table>	VALUE	MEANING	0	Synchronous process	1	Asynchronous process without result code	2	Asynchronous process with result code
VALUE	MEANING								
0	Synchronous process								
1	Asynchronous process without result code								
2	Asynchronous process with result code								
<i>pArg</i>	(const unsigned char LSFAR *) points to a set of ASCIIZ strings containing the arguments of the file to be run.								
<i>envpointer</i>	(const unsigned char LSFAR *) points to an ASCIIZ string that is not NULL and specifies the environment for the file to be run.								
<i>returncodes</i>	(unsigned char LSFAR *) points to an OS/2 data structure containing the return codes resulting from the file execution. This is the same data structure that is used with the OS/2 DosExecPgm function. (For more information about the return codes and the DosExecPgm function, see the <i>OS/2 Technical Library, Programming Guide Volume 1.</i>)								
<i>pgmPointer</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing only the name and extension of the file to be run.								
<i>reserved2</i>	(unsigned char LSFAR *) is a reserved pointer with the value 0.								
<i>remexecflags</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the remote executable flags that control program execution. This parameter is defined as follows:								

BIT	MEANING	SYMBOLIC CONSTANT
0	REM_PIPE_MODE	If 0, a message mode pipe is used input. If 1, a character mode pip standard input.
1	REM_WAIT_MODE	If 0, the OS/2 DosCWait function child process to finish before re 1, DosCWait waits for all spawned finish before returning.
2	REM_SIGL_MODE	If 0, map SIGINTR and SIGBREAK to remotng standard signals. If 1, signals as received.
3-15		Reserved, with a value of 0.

For more information about OS/2 signals, see the *OS/2 Technical Library, Programming Guide Volume 1.*

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_InternalError	2140	An internal error has occurred.
NERR_RunSrvPaused	2385	The run server you requested using the NET RUN command is paused.
NERR_ErrCommRunSrv	2389	An error occurred when communicating with a run server.
NERR_ErrConnRunSrv	2390	An error occurred when connecting to a run server.
NERR_ErrorExecingGhost	2391	An error occurred when starting a background process.
NERR_ShareNotFound	2392	The shared resource you are connected to could not be found.
NERR_PgmNotFound	2394	The program was not found.

Remarks

The NetRemoteExec function is a network extension of the OS/2 DosExecPgm function.

The executed process is run on the computer connected to the current drive of the caller. If the current drive of the caller is on a remote server, the child process is run on that server. If the current drive of the caller is a local drive, the child process is run locally.

The NetRemoteExec function requires that a remotely run process inherit one of the following handles:

HANDLE	MEANING
0	Standard input (stdin)
1	Standard output (stdout)
2	Standard error (stderr)

When this API initiates an asynchronous process, the process identification (PID) returned as the first word in the OS/2 data structure (pointed to by the *returncodes* parameter) is a valid local PID that represents the remote program. The PID can be passed to the OS/2 DosFlagProcess function to:

- Send signals to the remote process
- Call the OS/2 DosCWait function to wait for the remote process to exit
- Call the OS/2 DosKillProcess function to end the process

Related Information

For information about:

- Listing resources of a server, see [Share - NetShareEnum](#).
- Running a program, see "OS/2 DosExecPgm" in the *OS/2 Technical Library, Programming Guide Volume 1*.
- OS/2 DosCWait, see "OS/2 DosCWait" in the *OS/2 Technical Library, Programming Guide Volume 1*.

NetRemoteMove or Net32RemoteMove

NetRemoteMove or Net32RemoteMove

The NetRemoteMove API moves one or more files from one location to another on a server.

Restrictions

This API can be called from OS/2 workstations, but both the source and destination drives must be redirected drives. (See the *sourcepath* and *destpath* parameters for a description.)

Syntax

```
#include <netcons.h>
#include <remutil.h>

NetRemoteMove(sourcepath, destpath,
              sourcepass, destpass,
              openflags, moveflags,
              buf, usBuflen);          /* 16 bit */

Net32RemoteMove(sourcepath, destpath,
                sourcepass, destpass,
                openflags, moveflags,
                buf, ulBuflen);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sourcepath (const unsigned char LSFAR *) points to an ASCIIZ string containing the path name of the file to be moved. Wildcards can be used. This parameter must begin with either a redirected drive or a universal naming convention (UNC) name.

destpath (const unsigned char LSFAR *) points to an ASCIIZ string containing the path name to which *sourcepath* is to be moved. For a wildcard *sourcepath*, *destpath* must be a directory. This parameter must begin with either a redirected drive or a UNC name.

sourcepass (const unsigned char LSFAR *) is reserved and must be NULL.

destpass (const unsigned char LSFAR *) is reserved and must be NULL.

openflags (16-bit unsigned short or 32-bit unsigned long) specifies how *destpath* will be opened. This parameter is defined as follows:

BIT	MEANING
0-1	Used if <i>destpath</i> exists. If 0, the open fails; if 1, appended; and if 2, the file is overwritten.
2-3	Reserved, with a value of 0.
4	Used if <i>destpath</i> does not exist. If 0, the open fails; file is created.
5-15	Reserved, with a value of 0.

moveflags (16-bit unsigned short or 32-bit unsigned long) establishes control for the file move. This

parameter is defined as follows:

BIT	MEANING
0	If 1, <code>destpath</code> must be a file, and bit 1 must be 0.
1	If 1, <code>destpath</code> must be a directory, and bit 0 must be 0.
2-15	Reserved; the value of these bits must be 0.

buf (unsigned char LSFAR *) points to the *move_info* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NO_MORE_FILES	18	No more files are available.
ERROR_SHARING_VIOLATION	32	A sharing violation occurred.
ERROR_FILE_EXISTS	80	The file already exists.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_BadSource	2381	The source path is not valid.
NERR_BadDest	2382	The destination path is not valid.
NERR_DifferentServers	2383	The source and destination paths are on different servers.

Remarks

If the source and destination files are in the same directory, NetRemoteMove renames the source file. When the source and destination are on different drives, NetRemoteMove moves *sourcepath* to *destpath* and then deletes *sourcepath*.

The source and destination path names (*sourcepath* and *destpath*) supplied to the NetRemoteMove function must be on the same server. The following cases are valid:

- The source and destination are both files. The source file is copied to the destination file, subject to *openflags* and *moveflags* limitations.
- The source is either a file or wildcard, and the destination is a directory. The source files are copied to the destination directory, subject to *openflags* and *moveflags* limitations.

Related Information

For information about:

- Copying a file from one network location to another, see [Remote Utility - NetRemoteCopy](#).
- Determining whether a drive letter is local or redirected to a remote server, see [Use - NetUseGetInfo](#).
- Listing available resources on a server, see [Share - NetShareEnum](#).

NetRemoteTOD or Net32RemoteTOD

NetRemoteTOD or Net32RemoteTOD

The NetRemoteTOD API returns the time of day on a server.

Restrictions

This API can be called from DLS and OS/2 workstations.

Syntax

```
#include <netcons.h>
#include <remutil.h>

NetRemoteTOD(pszServername, buf,
             usBuflen);          /* 16 bit */

Net32RemoteTOD(pszServername, buf,
               ulBuflen);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

buf (unsigned char LSFAR *) points to the *time_of_day_info* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarte	2138	The Requester service has not been started.
NERR_InvalidCompute	2351	The specified computer name is not valid.

Related Information

For information about listing the servers on a network, see [Server - NetServerEnum2](#).

Requester Category

This category includes the following APIs:

[Requester - NetWkstaGetInfo](#)
[Requester - NetWkstaSetInfo](#)

Requester APIs control the operation of requesters. They are used with the ACCESS.H, NETCONS.H, and WKSTA.H header files.

The Requester functions enable applications to control the configuration of a requester. To configure a requester, an application calls the NetWkstaSetInfo API. The NetWkstaGetInfo API returns information about the configuration of a requester.

Note: The domain name of the requester (the *wki0_langroup* component in the *wksta_info* data structure) must not duplicate any computer name or user name on the network.

If both a computer and a domain have the same name, only one of them can run at a time. If the computer is started first, no other computer on the domain of the same name can be started. If any computer in the domain is started first, the computer with the domain's name cannot be started. For more information about domains, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.

DOS Considerations

Certain parameters are not used under DOS and, therefore, cannot be set. However, validity checks are done on most of the unused parameters for future expansion.

The following table lists the fields of the *wksta_info_x* data structure that can be set with the *parmnum* parameter. The **Used?** column indicates whether a component is used by DLS. (These fields are defined in [Requester Data Structures](#).)

PARAMETER	USED?
charwait	Yes
chartime	Yes
charcount	Yes
errlogsz	No
printbuftime	Yes
wrkheuristics	No

Requester Data Structures

For the defaults and ranges of these structured information fields, see the IBMLAN.INI parameter descriptions in the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

Using any ASCII text editor, you can view these structures in the \IBMLAN\NETSRC\H\WKSTA.H header file.

The NetWkstaGetInfo (level 0, 1, 2, 10 and 11) and NetWkstaSetInfo (level 0 and 1) APIs accept or return data at the specified level of detail, using the *wksta_info* data structure.

Requester Level 0

```
struct wksta_info_0 {
    unsigned short      wki0_reserved_1;
    unsigned long       wki0_reserved_2;
    unsigned char LSFAR * LSPTR wki0_root;
    unsigned char LSFAR * LSPTR wki0_computername;
    unsigned char LSFAR * LSPTR wki0_username;
    unsigned char LSFAR * LSPTR wki0_langgroup;
    unsigned char       wki0_ver_major;
    unsigned char       wki0_ver_minor;
    unsigned long       wki0_reserved_3;
    unsigned short      wki0_charwait;
    unsigned long       wki0_chartime;
    unsigned short      wki0_charcount;
    unsigned short      wki0_reserved_4;
    unsigned short      wki0_reserved_5;
    unsigned short      wki0_keeppconn;
    unsigned short      wki0_keeppsearch;
    unsigned short      wki0_maxthreads;
    unsigned short      wki0_maxcmds;
    unsigned short      wki0_reserved_6;
    unsigned short      wki0_numworkbuf;
    unsigned short      wki0_sizworkbuf;
    unsigned short      wki0_maxwrkcache;
    unsigned short      wki0_sesstimeout;
    unsigned short      wki0_sizerror;
    unsigned short      wki0_numalerts;
    unsigned short      wki0_numservices;
    unsigned short      wki0_errlogsz;
    unsigned short      wki0_printbuftime;
    unsigned short      wki0_numcharbuf;
    unsigned short      wki0_sizcharbuf;
    unsigned char LSFAR * LSPTR wki0_logon_server;
    unsigned char LSFAR * LSPTR wki0_wrkheuristics;
    unsigned short      wki0_mailslots;
};
```

where:

- *wki0_reserved_1*, *wki0_reserved_2*, *wki0_reserved_3*, *wki0_reserved_4*, *wki0_reserved_5*, and *wki0_reserved_6* are reserved and must be 0.
- *wki0_root* points to an ASCIIZ string containing the path to the IBMLAN directory of the computer. (The recommended default path is \IBMLAN.)
- *wki0_computername* points to an ASCIIZ string containing the computer name of the local requester that is being configured.
- *wki0_username* points to an ASCIIZ string containing the name of the user who is logged on to the requester.
- *wki0_langgroup* points to an ASCIIZ string containing the name of the domain to which the requester belongs.
- *wki0_ver_major* specifies the major version number of the LAN Server software running on the computer.
- *wki0_ver_minor* specifies the minor version number of the LAN Server software running on the computer.
- *wki0_charwait* indicates the number of seconds the requester waits for a remote serial or communication device to become available.
- *wki0_chartime* indicates the number of milliseconds the requester waits to collect data to send to a remote serial or communication device.
- *wki0_charcount* indicates the number of bytes of information the requester sends to a remote serial or communication device.
- *wki0_keeppconn* indicates the number of seconds an inactive connection from the requester to a resource of a server is maintained.

- *wki0_keepsearch* indicates the number of seconds an inactive search continues.
- *wki0_maxthreads* indicates the number of threads the requester can dedicate to the network.
- *wki0_maxcmds* indicates the number of simultaneous network device driver commands that can be sent to the network.
- *wki0_numworkbuf* indicates the number of internal buffers the requester has.
- *wki0_sizworkbuf* indicates the size (in bytes) of each internal buffer.

Use the following formula to determine the maximum values for both the *numworkbuf* and the *sizworkbuf* parameters:

$$(\text{sizworkbuf} + 268) * \text{numworkbuf} \leq 65515$$

The following values apply to both the *numworkbuf* and *sizworkbuf* parameters:

Default value:	4096
Minimum value:	1024
Maximum value:	16384

- *wki0_maxwrkcache* indicates the maximum size (in bytes) of an internal cache buffer.
- *wki0_sesstimeout* indicates the number of seconds before an inactive session between a requester and a server is discontinued.
- *wki0_sizerror* indicates the size (in bytes) of an internal error buffer.
- *wki0_numalerts* indicates the maximum number of clients that can receive alert messages.

Note: Each mailslot or semaphore registered by the NetAlertStart API is a different client, and the Alerter service registers at least three clients when it begins to run. For more information about alerts, see [Alert Category](#).

- *wki0_numservices* indicates the number of services that can be started on the requester at any time. For more information, see [Services Category](#).
- *wki0_errlogsz* indicates the maximum size (in KB) of the error log file of the requester.
- *wki0_printbuftime* indicates the number of seconds closing print jobs are closed.
- *wki0_numcharbuf* indicates the number of character pipe buffers and device buffers the requester can have.
- *wki0_sizcharbuf* indicates the maximum size (in bytes) of a character pipe buffer and device buffer.
- *wki0_logon_server* points to an ASCIIZ string containing the name of the logon server for the user who is currently logged on. The *wki0_logon_server* field may be null if the user performed either a local or non-validated logon. This field is non-null if the user successfully logged on to a domain. The logon server can, potentially, be any Domain Controller or backup server in the logon domain.
- *wki0_wrkheuristics* points to an ASCIIZ string of flags used to control a requester's operation. The heuristics default to values that are optimal for most configurations and normally need not be changed.
- *wki0_mailslots* specifies whether mailslots are allowed. If the value is 0, mailslots are not supported on this machine, and Netlogon service will not start.

Requester Level 1

```
struct wksta_info_1 {
    unsigned short          wkil_reserved_1;
    unsigned long           wkil_reserved_2;
    unsigned char LSFAR * LSPTR wkil_root;
    unsigned char LSFAR * LSPTR wkil_computername;
    unsigned char LSFAR * LSPTR wkil_username;
}
```

```

unsigned char LSFAR * LSPTR   wkil_langroup;
unsigned char                wkil_ver_major;
unsigned char                wkil_ver_minor;
unsigned long                wkil_reserved_3;
unsigned short               wkil_charwait;
unsigned long                wkil_chartime;
unsigned short               wkil_charcount;
unsigned short               wkil_reserved_4;
unsigned short               wkil_reserved_5;
unsigned short               wkil_keepconn;
unsigned short               wkil_keepsearch;
unsigned short               wkil_maxthreads;
unsigned short               wkil_maxcmds;
unsigned short               wkil_reserved_6;
unsigned short               wkil_numworkbuf;
unsigned short               wkil_sizworkbuf;
unsigned short               wkil_maxwrkcache;
unsigned short               wkil_sesstimeout;
unsigned short               wkil_sizerror;
unsigned short               wkil_numalerts;
unsigned short               wkil_numservices;
unsigned short               wkil_errlogsz;
unsigned short               wkil_printbuftime;
unsigned short               wkil_numcharbuf;
unsigned short               wkil_sizcharbuf;
unsigned char LSFAR * LSPTR   wkil_logon_server;
unsigned char LSFAR * LSPTR   wkil_wrkheuristics;
unsigned short               wkil_mailslots;
unsigned char LSFAR * LSPTR   wkil_logon_domain;
unsigned char LSFAR * LSPTR   wkil_oth_domains;
unsigned short               wkil_numdgrambuf;
};

```

where:

- The first 33 fields in this data structure are identical to those in the previous level.
- *wki1_logon_domain* names the domain to which the user is logged on. If no one is logged on, it is returned as NULL. In Directory and Security Server (DSS) logon, there is no domain; therefore, resource domain broadcast address (RBA) is used. RBA is the equivalent of a domain name and can be used as such in other API calls.
- *wki1_oth_domain* is an ASCIIZ string listing all domains on which the machine currently is listed. This is a *far* pointer to an ASCIIZ string that is a space-delimited list of domains. You can set *oth_domains* with NetWkstaSetInfo.
- *wkil_numdgrambuf* is the number of buffers allocated for receiving datagrams.

Requester Level 2

```

struct wksta_info_2 {
    unsigned short           wki2_reserved_1;
    unsigned long            wki2_reserved_2;
    unsigned char LSFAR * LSPTR wki2_root;
    unsigned char LSFAR * LSPTR wki2_computername;
    unsigned char LSFAR * LSPTR wki2_username;
    unsigned char LSFAR * LSPTR wki2_langroup;
    unsigned char            wki2_ver_major;
    unsigned char            wki2_ver_minor;
    unsigned long            wki2_reserved_3;
    unsigned short           wki2_charwait;
    unsigned long            wki2_chartime;
    unsigned short           wki2_charcount;
    unsigned short           wki2_reserved_4;
    unsigned short           wki2_reserved_5;
    unsigned short           wki2_keepconn;
    unsigned short           wki2_keepsearch;
    unsigned short           wki2_maxthreads;
};

```

```

unsigned short      wki2_maxcmds;
unsigned short      wki2_reserved_6;
unsigned short      wki2_numworkbuf;
unsigned short      wki2_sizworkbuf;
unsigned short      wki2_maxwrkcache;
unsigned short      wki2_sesstimeout;
unsigned short      wki2_sizerror;
unsigned short      wki2_numalerts;
unsigned short      wki2_numservices;
unsigned short      wki2_errlogsz;
unsigned short      wki2_printbuftime;
unsigned short      wki2_numcharbuf;
unsigned short      wki2_sizcharbuf;
unsigned char LSFAR * LSPTR wki2_logon_server;
unsigned char LSFAR * LSPTR wki2_wrkheuristics;
unsigned short      wki2_mailslots;
unsigned char LSFAR * LSPTR wki2_logon_domain;
unsigned char LSFAR * LSPTR wki2_oth_domains;
unsigned short      wki2_numdgrambuf;
unsigned char LSFAR * LSPTR wki2_resdom_name;
unsigned char LSFAR * LSPTR wki2_cell_name;
};

```

where:

- The first 35 fields in this data structure are identical to those in the previous level.
- *resdom_name* is the user's default resource domain. For LAN Server and OS/2 Warp Server logons, this value is set to NULL.
- *cell_name* is the name of the cell where the user is logged on.

Note: Logons invoked from the NetWkstaSetUID2 do not update fields during logon. This is consistent with the current implementation; LAN Server and DSS logons are only supported through the UPMEUSRL API.

Note: This level is only valid for DSS.

Requester Level 10

The *wksta_info_10* data structure is supplied for remote users who want to obtain certain information from a server. This data structure allows remote users to discover to which domain a server belongs.

Since a remote NetWkstaGetInfo at levels 0 and 1 requires administrative privilege, remote users who do not have privilege level ADMIN cannot use those structures. These levels provide the needed information for those users.

```

struct wksta_info_10 {
    unsigned char LSFAR * LSPTR wki10_computername;
    unsigned char LSFAR * LSPTR wki10_username;
    unsigned char LSFAR * LSPTR wki10_langroup;
    unsigned char      wki10_ver_major;
    unsigned char      wki10_ver_minor;
    unsigned char LSFAR * LSPTR wki10_logon_domain;
    unsigned char LSFAR * LSPTR wki10_oth_domains;
};

```

Requester Level 11

```

struct wksta_info_11 {
    unsigned char LSFAR * LSPTR wkill_computername;
    unsigned char LSFAR * LSPTR wkill_username;
    unsigned char LSFAR * LSPTR wkill_langroup;
    unsigned char      wkill_ver_major;
    unsigned char      wkill_ver_minor;
    unsigned char LSFAR * LSPTR wkill_logon_domain;
    unsigned char LSFAR * LSPTR wkill_oth_domains;
    unsigned char LSFAR * LSPTR wkill_resdom_name;
    unsigned char LSFAR * LSPTR wkill_cell_name;
};

```

Note: This level is only valid for DSS.

Related Information

For information about:

- Configuring requesters, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Using the IBMLAN.INI file, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.
- Using domains, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Using NCB architecture, see the *IBM Local Area Network Technical Reference*.
- Using *wksta_info_0* components, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetWkstaGetInfo or Net32WkstaGetInfo

NetWkstaGetInfo or Net32WkstaGetInfo

The NetWkstaGetInfo API returns configuration information about a requester.

Restrictions

This API can be called from DOS and OS/2 workstations. This API requires that only remote users have administrator authority for the level 0, 1, and 2 data structures. The level 10 and 11 data structure, however, can be obtained with either administrator or user authority.

Note: Levels 2 and 11 are valid only for DSS.

Syntax

```

#include <netcons.h>
#include <wksta.h>

NetWkstaGetInfo(pszServername, sLevel, buf,
               usBuflen, pusBytesAvail);          /* 16 bit */

Net32WkstaGetInfo(pszServername, ulLevel, buf,
                  ulBuflen, pulBytesAvail);      /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, 2, 10 or 11, specifying which data structure to use, as described in [Requester Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BadTransactConfig	2141	The server is not configured for transactions.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about changing the configuration of a local requester, see [Requester - NetWkstaSetInfo](#).

NetWkstaSetInfo or Net32WkstaSetInfo

NetWkstaSetInfo or Net32WkstaSetInfo

The NetWkstaSetInfo API configures a requester.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server. Administrator authority is required only for remote users to call this API. No access authority is required for local execution.

The fields *wki0_computername* and *wki0_langroup* cannot be set by users or administrators.

The values of the *wksta_info* data structure are separated by spaces. An empty list is valid. (As usual, a NULL pointer indicates that the field should not be changed.) An empty element, however, is not valid.

The NetWkstaSetInfo API rejects attempts to change *oth_domains* in the following cases:

- The name list was incorrect.
- One of the names could not be added to the network adapters managed by LAN Server.

Syntax

```
#include <netcons.h>
```



```
#include <wksta.h>

NetWkstaSetInfo(pszServername, sLevel, buf,
               usBuflen, parmnum);           /* 16 bit */

Net32WkstaSetInfo(pszServername, ulLevel, buf,
                 ulBuflen, parmnum);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Requester Data Structures](#).

parmnum (16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a specific field in the data structure is to be passed. If the value is 0, the entire data structure is sent. Otherwise, *parmnum* specifies which field in the data structure is to be sent. If not 0, *parmnum* can be any one of the following:

SYMBOLIC CONSTANT	VALUE	FIELD TO BE PASSED
WKSTA_CHARWAIT_PARMNUM	10	wkiX_charwait
WKSTA_CHARTIME_PARMNUM	11	wkiX_chartime
WKSTA_CHARCOUNT_PARMNUM	12	wkiX_charcount
WKSTA_ERRLOGSZ_PARMNUM	27	wkiX_errlogsz
WKSTA_PRINTBUFTIME_PARMNUM	28	wkiX_printbuftime
WKSTA_WRKHEURISTICS_PARMNUM	32	wkiX_wrkheuristics
WKSTA_OTHDOMAINS_PARMNUM	35	wkiY_oth_domains

NOTE :

X = 0 or 1
Y = 1

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.

NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosDevIOCtl
- DosFsCtl
- DosGetShrSeg

Related Information

For information about retrieving the configuration of a local requester, see [Requester - NetWkstaGetInfo](#).

RIPL Category

This category includes the following APIs:

[RIPL - NetCreateRIPLMachine](#)
[RIPL - NetDeleteRIPLMachine](#)
[RIPL - NetEnumRIPLMachine](#)
[RIPL - NetGetRIPLMachineInfo](#)
[RIPL - NetSetRIPLMachineInfo](#)

Remote IPL (RIPL) APIs can manage the definition of RIPL workstations at a RIPL server. These APIs use the RIPL.H header file.

RIPL Machine Data Structures

The *sLevel* parameter controls the level of information provided to or returned from the NetCreateRIPLMachine, NetDeleteRIPLMachine, NetEnumRIPLMachine, NetGetRIPLMachineInfo, and NetSetRIPLMachineInfo APIs. These APIs use a level 0, 1, 2, or 12 data structure.

RIPL Machine Information Level 0

```
struct ripl_machine_info_0 {
    unsigned char    rmi0_name[CNLEN+1];
};
```

where *rmi0_name* is an ASCIIZ string containing the name of the RIPL workstation. Any characters are valid except for imbedded blanks and the following:

" / \ [] : | < > + = ; , ? *

The workstation name must be unique within a domain.

RIPL Machine Information Level 1

```
struct ripl_machine_info_1 {
    unsigned char    rmi1_name[CNLEN+1];
    unsigned char LSFAR * LSPTR    rmi1_remark;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *rmi1_remark* points to an ASCIIZ string that contains descriptive information about the workstation. It can be up to 48 bytes long. This field is optional; a NULL pointer or string indicates that there is no remark.

RIPL Machine Information Level 2

```
struct ripl_machine_info_2 {
    unsigned char    rmi2_name[CNLEN+1];
    unsigned char LSFAR * LSPTR    rmi2_remark;
    unsigned char    rmi2_adapter_address[NNLEN+1];
    unsigned char    rmi2_pad_1;
    unsigned char    rmi2_OS2_boot_drive;
    unsigned char    rmi2_pad_2;
    unsigned char    rmi2_DOS_imageid[DOSIMGLEN+1];
    unsigned char    rmi2_pad_3;
    unsigned char LSFAR * LSPTR    rmi2_DOS_srv_rec_ID;
    unsigned char LSFAR * LSPTR    rmi2_OS2_srv_rec_ID;
    unsigned long     rmi2_OS2_config_flags;
};
```

where:

- The first two fields in this data structure are identical to those in the previous level.
- *rmi2_adapter_address* is an ASCIIZ string that contains the network adapter address.
- *rmi2_pad_1* word-aligns the data structure.
- *rmi2_OS2_boot_drive* indicates the RIPL workstation's boot drive. This member is valid only in (1) an OS/2 create operation, or

(2) a set operation before the workstation has been IPLed. A default value of *Z* is assumed if this member is NULL.

- *rmi2_pad_2* word-aligns the data structure.
- *rmi2_DOS_imageid* is an ASCIIZ string that contains the name of the image definition used to remote IPL a DOS requester.
- *rmi2_pad_3* word-aligns the data structure.
- *rmi2_DOS_srv_rec_ID* points to an ASCIIZ string that contains the name of the identifier of the server record that services the DOS requester's remote IPL request when the requester is started.
- *rmi2_OS2_srv_rec_ID* points to an ASCIIZ string that contains the name of the identifier of the server record that services the OS/2 requester's remote IPL request when the requester is started.

Note: The two preceding fields describe the DOS and OS/2 records that are added to RPL.MAP on the RIPL server. For more information, refer to the *IBM OS/2 LAN Server Version 4.0 Network Administrator Reference Volume 3: Network Administrator Tasks*.

- *rmi2_OS2_config_flags* contains bit flag mappings for the configuration of an OS/2 RIPL requester. This parameter is not used for a DOS RIPL requester. The fields are defined as follows:

SYMBOLIC CONSTANT	VALUE	BIT POSITION
OS2_SWAPPER_LOCAL	0x0000	0
OS2_SWAPPER_SERVER	0x0001	0
OS2_BUS_MCA	0x0000	1
OS2_BUS_ISA	0x0002	1
OS2_DISPLAY_IBMVGA32	0x0000	4
OS2_DISPLAY_IBMXGA32	0x0010	4
OS2_DISPLAY_IBM8514	0x0020	5
OS2_DISPLAY_SVGA	0x0040	6
OS2_DISPLAY_S3SVGA	0x0080	7
OS2_DISPLAY_IBMEGA	0x0100	8
OS2_DISPLAY_PS55_16	0x8010	15 and 4
OS2_DISPLAY_PS55_256	0x8020	15 and 5
OS2_DISPLAY_PS55_256P	0x8040	15 and 6
OS2_KEYBOARD_101	0x0000	12
OS2_KEYBOARD_DBCS_103	0x1000	12
OS2_KEYBOARD_DBCS_106	0x2000	13
OS2_KEYBOARD_DBCS_AX	0x4000	14

Remarks

The information in *machine_info_2* is stored as follows:

MEMBER	FIELD	COMMENT
rmi2_name	2	
rmi2_remark	N/A	NET.ACC, as part of user account.
rmi2_adapter_address	1	
rmi2_OS2_boot_drive	6	OS/2 only.
rmi2_DOS_imageid	4	DOS only.
rmi2_DOS_srv_rec_ID	C	See the note at the end of this table.
rmi2_OS2_srv_rec_ID	C	See the note at the end of this table.

NOTE: In RPL.MAP, workstation record field C refers to a server record ID (which also is field C in the server record section of RPL.MAP). Specify either a DOS server record ID or an OS/2 server record ID, but not both.

RIPL Machine Information Level 12

```
struct ripl_machine_info_12 {
    unsigned char    rmi12_name[CNLEN+1];
    unsigned char LSFAR * LSPTR rmi12_remark;
    unsigned char    rmi12_adapter_address[NNLEN+1];
    unsigned char    rmi12_pad_1;
    unsigned char    rmi12_model_name[CNLEN+1];
}
```

where:

- The first four fields in this data structure are identical to those in the previous level.
- *rmi12_model_name* is an ASCIIZ string that contains the name of an existing RIPL client to be used as a model when creating a new (clone) RIPL client.

NetCreateRIPLMachine or Net32CreateRIPLMachine

NetCreateRIPLMachine or Net32CreateRIPLMachine

The NetCreateRIPLMachine API creates a RIPL workstation definition.

Restrictions

This API requires the user have administrator privilege. If a session already is active between a workstation and its RIPL server, this API does not function at level 12.

Syntax

```
#include <ripl.h>

NetCreateRIPLMachine(pszServername, sLevel, buf,
                    usBuflen);    /* 16 bit */

Net32CreateRIPLMachine(pszServername, ulLevel, buf,
                      ulBuflen);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit unsigned short or 32-bit unsigned long) must be 2 or 12, specifying which level of data structure to use, as described in [RIPL Machine Information Level 2](#) and RIPL Machine Information Level 12.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_RPL_MAPWriteError	5404	An error occurred while updating the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist on the remote IPL server.
NERR_FITSFileNotFound	5406	An error was detected while reading configuration files in the IBMLAN\RPL\FITS directory on the remote IPL server.
NERR_MACHINESFileNotFound	5407	An error was detected while reading configuration files in the IBMLAN\RPL\MACHINES directory tree on the remote IPL server.
NERR_IBMCOMFileNotFound	5408	An error was detected while reading configuration files in the IBMLAN\RPL\IBMCOM directory tree on the remote IPL server.
NERR_RPLUSERFileNotFound	5409	An error was detected while reading configuration files in the IBMLAN\RPLUSER directory tree on the remote IPL server.

NERR_RPLFileNotFound	5410	An error was detected while reading configuration files in the IBMLAN\RPL directory tree on the remote IPL server.
NERR_MachineFilesLocked	5411	Machine configuration files are locked by the remote IPL server file system.
NERR_ServerRecordInvalid	5412	The server record identifier is not defined in an enabled server record entry in the RPL.MAP file.
NERR_RemarkInvalid	5413	The machine definition remark was more than 48 bytes.
NERR_NDISProfileNotFound	5414	The network adapter directory profile could not be found in the NDISDD.PRO file on the remote IPL server.
NERR_MachNameTooLong	5415	Machine definition names for DOS or FAT file systems cannot exceed 8 bytes.
NERR_CreateMachNameInvalid	5420	The machine definition could not be created, because the name is not unique in the domain.
NERR_CreateMachNameExists	5421	The machine definition already exists on the remote IPL server.
NERR_CreateMachModelInvalid	5422	The model machine name does not exist on the remote IPL server.
NERR_CreateMachDriveInvalid	5424	The remote IPL OS/2 boot drive identifier is not valid.
NERR_NetworkAddressInvalid	5425	The network adapter address is not valid.
NERR_CreateMachAddrExists	5426	A machine definition already exists with the specified network adapter address.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_CreateMachReadError	5428	An error was detected while reading machine configuration files on the remote IPL server.
NERR_CreateMachWriteError	5429	An error was detected while writing machine configuration files on the remote IPL server.
NERR_CreateMachFileNotFound	5430	Files required for creating a machine definition could not be found on the remote IPL server.
NERR_CreateMachDirFailure	5432	An error was detected while creating machine configuration directories on the remote IPL server.
NERR_CreateMachACLFailure	5433	An error was detected while creating

		access control profiles on the remote IPL server.
NERR_DOSImageNotFound	5480	The DOS image file name does not exist.

NetDeleteRIPLMachine or Net32DeleteRIPLMachine

NetDeleteRIPLMachine or Net32DeleteRIPLMachine

The NetDeleteRIPLMachine API deletes a RIPL workstation definition.

Restrictions

This API requires the user have administrator privilege. If a session already is active between a workstation and its RIPL server, this API does not function.

Syntax

```
#include <ripl.h>

NetDeleteRIPLMachine(pszServername,
                    MachineName); /* 16 bit */

Net32DeleteRIPLMachine(pszServername,
                     MachineName); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

MachineName (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the RIPL workstation definition that is to be deleted.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_RPL_MAPWriteError	5404	An error occurred while updating the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist

		on the remote IPL server.
NERR_MachineFilesLocked	5411	Machine configuration files are locked by the remote IPL server file system.
NERR_MachNameTooLong	5415	Machine definition names for DOS or FAT file systems cannot exceed 8 bytes.
NERR_DelMachFailure	5440	The machine definition could not be deleted.
NERR_DelMachDirFailure	5441	An error was detected while deleting machine configuration directories on the remote IPL server.

NetEnumRIPLMachine or Net32EnumRIPLMachine

NetEnumRIPLMachine or Net32EnumRIPLMachine

The NetEnumRIPLMachine API lists RIPL workstations of the specified types on a server.

Restrictions

A process with any access authority can call this API with a level of 0 or 1. To call this API with a level of 2, the calling process must have administrator privilege or server operator privilege.

Syntax

```
#include <ripl.h>

NetEnumRIPLMachine(pszServername, sLevel, type,
                  buf, usBuflen, pusEntriesReturned,
                  pusEntriesAvail);          /* 16 bit */

Net32EnumRIPLMachine(pszServername, ulLevel, type,
                   buf, ulBuflen, pulEntriesReturned,
                   pulEntriesAvail);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit unsigned short or 32-bit unsigned long) must be 0, 1, or 2, specifying the level of detail for the *ripl_machine_info* structure, as described in [RIPL Machine Data Structures](#).

type (16-bit unsigned short or 32-bit unsigned long) indicates the types of RIPL workstation definitions that are to be returned in *buf*. The following types are defined in RIPL.H:

SYMBOLIC CONSTANT	BIT MASK
MTYPE_OS2_RIPL_REQ	0x0001
MTYPE_DOS_RIPL_REQ	0x0002

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist on the remote IPL server.
NERR_EnumMachNotFound	5450	A machine definition of the type requested does not exist on the remote IPL server.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about listing all servers (members of group SERVERS) or all OS/2 requesters (members of group RPLGROUP) in a domain, see [Group - NetGroupGetUsers](#).

NetGetRIPLMachineInfo or Net32GetRIPLMachineInfo

NetGetRIPLMachineInfo or Net32GetRIPLMachineInfo

The NetGetRIPLMachineInfo API retrieves information about a RIPL workstation.

Restrictions

A process with any access authority can call this API with a level of 0 or 1. To call this API with a level of 2, the calling process must have administrator privilege or server operator privilege.

Syntax

```
#include <ripl.h>

NetGetRIPLMachineInfo(pszServername,
                     MachineName, sLevel, buf,
                     usBuflen, pusBytesAvail); /* 16 bit */

Net32GetRIPLMachineInfo(pszServername,
                       MachineName, ulLevel, buf,
                       ulBuflen, pulBytesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>MachineName</i>	(unsigned char LSFAR *) points to an ASCIIZ string containing the name of the RIPL workstation definition for which information is to be retrieved. This pointer cannot be NULL.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit unsigned short or 32-bit unsigned long) must be 0, 1, or 2, specifying the level of detail returned in the <i>ripl_machine_info</i> structure as described in RIPL Machine Data Structures . Level 12 is not valid for this API.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist on the remote IPL server.
NERR_MACHINESFileNotFound	5407	An error was detected while reading configuration files in the IBMLAN\RPL\MACHINES directory tree on the remote IPL server.
NERR_RPLUSERFileNotFound	5409	An error was detected while reading configuration files in the IBMLAN\RPLUSER directory tree on the remote IPL server.
NERR_RPLFileNotFound	5410	An error was detected while reading configuration files in the IBMLAN\RPL directory tree on the remote IPL server.
NERR_MachineFilesLocked	5411	Machine configuration files are locked by the remote IPL server file system.
NERR_MachNameTooLong	5415	Machine definition names for DOS or FAT file systems cannot exceed 8 bytes.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

NetSetRIPLMachineInfo or Net32SetRIPLMachineInfo

NetSetRIPLMachineInfo or Net32SetRIPLMachineInfo

The NetSetRIPLMachineInfo API changes information about a RIPL workstation.

Restrictions

- Administrator authority is required for this API except when a user issues a call concerning the user.
- This API requires either administrator privileges or server operator privilege.
- If the operating system for a workstation is to be changed, the entire level 2 structure must be used (*parmnum*=0).

Syntax

```
#include <ripl.h>

NetSetRIPLMachineInfo(pszServername,
    MachineName, sLevel, buf,
    usBuflen, parmnum);    /* 16 bit */

Net32SetRIPLMachineInfo(pszServername,
    MachineName, ulLevel, buf,
    ulBuflen, parmnum);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- MachineName* (unsigned char LSFAR *) points to an ASCIIZ string containing the name of the RIPL workstation definition for which information is to be changed. This pointer cannot be NULL.
- sLevel* or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1 or 2, specifying the level of detail returned in the *ripl_machine_info* structure as described in [RIPL Machine Information Level 1](#) and [RIPL Machine Information Level 2](#).
- buf* (unsigned char LSFAR *) points to a *ripl_machine_info* structure if *parmnum* is 0. Otherwise, *buf* points to the field of the data structure that will be changed. This pointer cannot be NULL.
- parmnum* (16-bit unsigned short or 32-bit unsigned long) determines whether the buffer contains a complete *ripl_machine_info* structure or a single field. If the value is 0, *buf* must contain a complete *ripl_machine_info* structure. Otherwise, *parmnum* must specify the ordinal position for a field in the data structure, defined in RIPL.H:

SYMBOLIC CONSTANT	VALUE	DATA STRUCTURE FIELD
RIPL_MACH_REMARK_PARMNUM	2	rmiXremark
RIPL_MACH_ADAPTER_PARMNUM	3	rmi2_adapter_address
RIPL_MACH_OS2_BOOTDRIVE_PARMNUM	5	rmi2_OS2_boot_drive
RIPL_MACH_DOS_IMAGEID_PARMNUM	7	rmi2_DOS_imageid
RIPL_MACH_DOS_SRVRECID_PARMNUM	9	rmi2_DOS_srv_rec_ID
RIPL_MACH_OS2_SRVRECID_PARMNUM	10	rmi2_OS2_srv_rec_ID
NOTE: X = 1 or 2.		

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_RPL_MAPWriteError	5404	An error occurred while updating the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist on the remote IPL server.
NERR_FITSFileNotFound	5406	An error was detected while reading configuration files in the IBMLAN\RPL\FITS directory on the remote IPL server.
NERR_MACHINESFileNotFound	5407	An error was detected while reading configuration files in the IBMLAN\RPL\MACHINES directory tree on the remote IPL server.
NERR_IBMCOMFileNotFound	5408	An error was detected while reading configuration files in the IBMLAN\RPL\IBMCOM directory tree on the remote IPL server.
NERR_RPLUSERFileNotFound	5409	An error was detected while reading configuration files in the IBMLAN\RPLUSER directory tree on the remote IPL server.
NERR_RPLFileNotFound	5410	An error was detected while reading configuration files in the IBMLAN\RPL directory tree on the remote IPL server.
NERR_MachineFilesLocked	5411	Machine configuration files are locked by the remote IPL server file system.
NERR_ServerRecordInvalid	5412	The server record identifier is not defined in an enabled server record entry in the RPL.MAP file.
NERR_RemarkInvalid	5413	The machine definition remark was more than 48 bytes.
NERR_NDISProfileNotFound	5414	The network adapter directory profile

		could not be found in the NDISDD.PRO file on the remote IPL server.
NERR_MachNameTooLong	5415	Machine definition names for DOS or FAT file systems cannot exceed 8 bytes.
NERR_CreateMachDriveInvalid	5424	The remote IPL OS/2 boot drive identifier is not valid.
NERR_NetworkAddressInvalid	5425	The network adapter address is not valid.
NERR_CreateMachAddrExists	5426	A machine definition already exists with the specified network adapter address.
NERR_CreateMachReadError	5428	An error was detected while reading machine configuration files on the remote IPL server.
NERR_CreateMachWriteError	5429	An error was detected while writing machine configuration files on the remote IPL server.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_CreateMachFileNotFound	5430	Files required for creating a machine definition could not be found on the remote IPL server.
NERR_CreateMachDirFailure	5432	An error was detected while creating machine configuration directories on the remote IPL server.
NERR_CreateMachACLFailure	5433	An error was detected while creating access control profiles on the remote IPL server.
NERR_SetMachDriveFailure	5470	The remote IPL OS/2 boot drive identifier cannot be changed because the machine has a Workplace Shell desktop defined.
NERR_SetMachNameInvalid	5471	The machine definition name cannot be changed.
NERR_SetMachVersionInvalid	5472	The machine definition OS/2 version cannot be changed.
NERR_SetMachCompInvalid	5473	The machine definition parameter specified is not valid for the current machine definition type.
NERR_DOSImageNotFound	5480	The DOS image file name does not exist.

Serial Device Category

This category includes the following APIs:

[Serial Device - NetCharDevControl](#)
[Serial Device - NetCharDevEnum](#)
[Serial Device - NetCharDevGetInfo](#)
[Serial Device - NetCharDevQEnum](#)
[Serial Device - NetCharDevQGetInfo](#)
[Serial Device - NetCharDevQPurge](#)
[Serial Device - NetCharDevQPurgeSelf](#)
[Serial Device - NetCharDevQSetInfo](#)

Serial device APIs control shared serial devices and their associated queues.

For an application to communicate with a device such as a serial printer or a plotter, the application must be able to communicate directly and interactively. The communication must allow commands to be submitted dynamically and protocols to be changed as the application runs. The LAN Server software defines these types of communication devices as *serial devices*. This definition is not limited to devices attached to hardware serial ports.

The serial device APIs are used with the CHARDEV.H and NETCONS.H header files.

The LAN Server software can pool serial devices of the same type into a serial device queue to which a requesting application makes its connection. A *serial device queue* can contain one or more serial devices and simultaneously allow multiple applications to connect to one of the available serial devices. Serial device queues can pool serial devices only on a server.

Note: Serial device queues exist only while they are being shared. In contrast, a spooled device queue (such as for a printer) exists from the time it is created (by calling the appropriate Add function) to the time it is removed.

Before an application can communicate with a serial device, the following must occur:

- The server must have a serial device connected to one of its available LPT or COM ports.
- A serial device queue must be created and shared on the network.
- A requesting application must (1) explicitly redirect a local or null device name to the shared serial device queue by calling NetUseAdd, or (2) implicitly open the serial device queue by calling DosOpen and specifying the name of a queue.

An explicit connection allows the application to refer to the serial device queue with a local device name. An explicit connection does not, however, open a serial device. For more information about redirecting a local device name to a shared resource, see [Use Category](#).

To illustrate how serial devices and queues work on the LAN, consider the following scenario. Assume that there are 4 serial devices connected to the communication ports of a server in the following manner:

PORT	DEVICE
COM1, COM2, COM3	Printers
COM4	Special

Once the serial devices are connected to the ports of a server, an application can create a serial device queue by calling the NetShareAdd function and specifying the share type as a serial device queue. In this scenario, assume the NetShareAdd function is called three times, creating the following 3 queues:

QUEUE NAME	PRIORITY	DEVICE NAME
SPLQ	5	COM4
PRINTQ	5	COM1, COM2, COM3
VIPPRINT	1	COM3

The COM3 port is allocated for use by 2 different queues, PRINTQ and VIPPRINT. After calling the NetShareAdd function to create a queue, you can set a parameter to assign a priority to the queue by calling NetCharDevQSetInfo. The priority can be from 1 (high) through 9 (low). Generally, the LAN Server software allows requests to serial device queues with a higher priority to access the pool of serial devices before other queues with lower priorities.

At this point, an application can connect to the shared serial device queue and begin communicating with one of the pooled serial devices in the queue.

If more than 1 serial device happens to be available in a serial device queue, the queue returns the first available serial device to the requesting application. If no devices currently are available, the queue puts the request on a waiting list until a serial device becomes available. The queue waits only as long as the *charwait* parameter of a requester specifies. If the thread undergoes a timeout while waiting for a serial device to become available, the DosOpen function returns the ERROR_BAD_NET_RESP error code.

An application can check (1) a particular serial device to see if it is working by calling the NetCharDevGetInfo function, or (2) all devices by calling the NetCharDevEnum function. An application also can check to see if the queue is busy or where the request of the application is on the queue waiting list by calling the NetCharDevQGetInfo function. To check all queues, call the NetCharDevQEnum function.

Applications can call the NetCharDevQPurgeSelf function to eliminate all requests submitted to a particular serial device queue from the requester of that application. To remove all requests submitted by all applications from the queue, call NetCharDevQPurge. A process that currently has a device open is unaffected.

When the application no longer needs the device, it should call the DosClose function to return control of the serial device to the serial device queue, allowing another application to use it. If the application cannot call the DosClose function successfully to close the serial device queue, the application can call the NetCharDevControl function to force the serial device queue closed.

When an application successfully opens a remote serial device, the values of the *chartime* and *charcount* components from the *wksta_info* data structure control how information flows across the network to other pertinent requesters and servers. Any application changing these values on the requester where the open was performed should note the following:

- The *chartime* and *charcount* components affect all applications running on the requester.
- Network efficiency, network response time, and network throughput might be slowed.

Serial Device Data Structures

The *sLevel* parameter controls the level of information provided to or returned from the data structures used by the following functions:

- NetCharDevEnum
- NetCharDevGetInfo
- NetCharDevQEnum
- NetCharDevQGetInfo
- NetCharDevQSetInfo

Serial Device Level 0

The NetCharDevEnum and NetCharDevGetInfo functions use the following data structure when the *sLevel* parameter is 0:

```
struct chardev_info_0 {
    unsigned char ch0_dev[DEVLEN+1];
};
```

where *ch0_dev* is an ASCIIZ string containing the device name associated with the serial device.

Serial Device Level 1

The NetCharDevEnum and NetCharDevGetInfo functions use the following data structure when the *sLevel* parameter is 1:

```
struct chardev_info_1 {
    unsigned char      ch1_dev[DEVLEN+1];
```



```

unsigned char      ch1_pad1;
unsigned short     ch1_status;
unsigned char      ch1_username[UNLEN+1];
unsigned char      ch1_pad2;
unsigned long      ch1_time;
};

```

where:

- *ch1_dev* is an ASCIIZ string specifying the device name associated with the serial device.
- *ch1_pad1* word-aligns the data structure components.
- *ch1_status* specifies the status of the device. *ch1_status* is defined as follows:

BIT	MEANING
0	Reserved, with a value of 0.
1	If 0, the device is idle; if 1, the device is open and presumably in use by some application.
2	If 0, the device has encountered no errors; if 1, the device has encountered an error.
3-15	Reserved, with a value of 0.

- *ch1_username* is an ASCIIZ string specifying the current user of the device.
- *ch1_pad2* word-aligns the data structure components.
- *ch1_time* specifies the number of seconds the current application has been connected to the serial device.

Serial Device Queue Level 0

The NetCharDevQEnum, NetCharDevQGetInfo, and NetCharDevQSetInfo functions use the following data structure when the *sLevel* parameter is 0:

```

struct chardevQ_info_0 {
    unsigned char cq0_dev[NNLEN+1];
};

```

where *cq0_dev* is an ASCIIZ string containing the queue name for the serial device queue.

Serial Device Queue Level 1

The NetCharDevQEnum, NetCharDevQGetInfo, and NetCharDevQSetInfo functions use the following data structure when the *sLevel* parameter is 1:

```

struct chardevQ_info_1 {
    unsigned char      cq1_dev[NNLEN+1];
    unsigned char      cq1_pad;
    unsigned short     cq1_priority;
    unsigned unsigned char LSFAR * LSPTR    cq1_devs;
    unsigned short     cq1_numusers;
    unsigned short     cq1_numahead;
};

```

```
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *cq1_pad* word-aligns the data structure components.
- *cq1_priority* specifies the queue priority. *cq1_priority* can be from 1 (highest priority) through 9 (lowest priority).
- *cq1_devs* points to an ASCIIZ string containing the device names assigned to the queue (such as COM1 COM3).
- *cq1_numusers* specifies the number of user names in the queue.
- *cq1_numahead* specifies the number of user names in front of a particular user. To find the number of users, specify *pszUserID* with either the NetCharDevQEnum or the NetCharDevQGetInfo function. If *cq1_numahead* is -1, then *pszUserID* is not currently in the queue.

NetCharDevControl or Net32CharDevControl

NetCharDevControl or Net32CharDevControl

The NetCharDevControl API forces a serial device closed.

Normally, a serial device is closed with a call to the DosClose function. If an opened device cannot be closed with DosClose, NetCharDevControl can be called to force the device closed.

Restrictions

This API can be called from DLS, and OS/2 workstations, but only DLS can issue this call to a remote LAN Server workstation. Users with administrator or comm operator privilege can call this API.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevControl(pszServername, devname, operation);    /* 16 bit */
Net32CharDevControl(pszServername, devname, operation); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

devname (const unsigned char LSFAR *) points to an ASCIIZ string specifying which device to change.

operation (16-bit short or 32-bit unsigned long) specifies the operation, defined in CHARDEV.H as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
CHARDEV_CLOSE	0	Closes the serial device

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of

return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_DevInvalidOpCode	2331	The operation is not valid for this device.
NERR_DevNotFound	2332	This device cannot be shared.
NERR_DevNotOpen	2333	This device was not open.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about:

- Changing a serial device queue, see [Serial Device - NetCharDevQSetInfo](#).

- Retrieving information about a serial device, see [Serial Device - NetCharDevGetInfo](#).
- Retrieving information about a serial device queue, see [Serial Device - NetCharDevQGetInfo](#).

NetCharDevEnum or Net32CharDevEnum

NetCharDevEnum or Net32CharDevEnum

The NetCharDevEnum API provides information about all available serial devices pooled in shared serial device queues on a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevEnum(pszServername, sLevel, buf,
               usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32CharDevEnum(pszServername, ulLevel, buf,
                 ulBuflen, pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Serial Device Level 0](#) and [Serial Device Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not

been started.

NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about listing serial device queues on a server, see [Serial Device - NetCharDevQEnum](#).

NetCharDevGetInfo or Net32CharDevGetInfo

NetCharDevGetInfo or Net32CharDevGetInfo

The NetCharDevGetInfo API retrieves information about a particular serial device in a shared serial device queue on a server.

A device can belong to more than one queue.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevGetInfo(pszServername, devname, sLevel,
                 buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32CharDevGetInfo(pszServername, devname, ulLevel,
```

```
buf, ulBuflen, pulBytesAvail);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

devname (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the device.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Serial Device Level 0](#) and [Serial Device Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_DevNotFound	2332	This device cannot be shared.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_InvalidComputer	2351	The specified computer name is not

valid.

Other codes could be returned from the following functions :

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Listing all serial device queues, see [Serial Device - NetCharDevEnum](#).
- Changing the state of a serial device, see [Serial Device - NetCharDevControl](#).

NetCharDevQEnum or Net32CharDevQEnum

NetCharDevQEnum or Net32CharDevQEnum

The NetCharDevQEnum API enumerates all serial device queues on a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevQEnum(pszServername, pszUserID, sLevel, buf,
               usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32CharDevQEnum(pszServername, pszUserID, ulLevel, buf,
                 ulBuflen, pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszUserID</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing a user name. To use <i>pszUserID</i> to calculate how many requests are pending ahead in the queue, examine the <i>cq1_numahead</i> field.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in Serial Device Queue Level 0 and Serial Device Queue Level 1 .

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of

return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Deleting the contents of a serial device queue, see [Serial Device - NetCharDevQPurge](#).
- Listing serial devices on a server, see [Serial Device - NetCharDevEnum](#).

NetCharDevQGetInfo or Net32CharDevQGetInfo

NetCharDevQGetInfo or Net32CharDevQGetInfo

The NetCharDevQGetInfo API retrieves information about a particular serial device queue on a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevQGetInfo(pszServername, pszQueueName, pszUserID, sLevel,
                  buf, usBuflen, pusBytesAvail);      /* 16 bit */

Net32CharDevQGetInfo(pszServername, pszQueueName, pszUserID, ulLevel,
                   buf, ulBuflen, pulBytesAvail);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- pszQueueName* (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of a particular printer queue.
- pszUserID* (const unsigned char LSFAR *) points to an ASCIIZ string containing a user name. To use *pszUserID* to calculate how many requests are pending ahead in the queue, examine the *cq1_numahead* field.
- sLevel* or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Serial Device Queue Level 0](#) and [Serial Device Queue Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.

NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.
NERR_NoCommDevs	2337	The user does not belong to this group.
NERR_QueueNotFound	2338	A queue does not exist for this request.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Listing serial device queues on a server, see [Serial Device - NetCharDevQEnum](#).
- Changing the state of a serial device queue, see [Serial Device - NetCharDevQSetInfo](#).

NetCharDevQPurge or Net32CharDevQPurge

NetCharDevQPurge or Net32CharDevQPurge

The NetCharDevQPurge API deletes all pending requests on a serial device queue.

This function deletes only requests that have not yet been assigned to a device. A process that currently has a device open is unaffected. All pending requests queued on *pszQueueName* are canceled, returning the ERROR_BAD_NET_RESP error code for each call to the DosOpen function. All handles still are valid.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevQPurge(pszServername, pszQueueName);      /* 16 bit */
Net32CharDevQPurge(pszServername, pszQueueName);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszQueueName (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of a particular printer queue.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_NoCommDevs	2337	There are no shared communication

devices.

NERR_QueueNotFound	2338	A queue does not exist for this request.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about:

- Closing the current session of a serial device, see [Serial Device - NetCharDevControl](#).
- Deleting the contents of a serial device queue, see [Serial Device - NetCharDevQPurgeSelf](#).
- Listing serial device queues on a server, see [Serial Device - NetCharDevQEnum](#).
- Changing the state of a serial device queue, see [Serial Device - NetCharDevQSetInfo](#).

NetCharDevQPurgeSelf or Net32CharDevQPurgeSelf

NetCharDevQPurgeSelf or Net32CharDevQPurgeSelf

The NetCharDevQPurgeSelf API deletes from a serial device queue all pending requests that were submitted by a particular computer.

The NetCharDevQPurgeSelf function deletes all requests that the NetCharDevQPurge function specifies, except that only requests from *computername* are deleted. A process that currently has a device open is unaffected.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS can issue this call to a remote LAN Server workstation. Administrator authority is required to delete requests from other computers when NetCharDevQPurgeSelf is called remotely. No administrator authority is required to delete a queue that was originated by the caller.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevQPurgeSelf(pszServername, pszQueueName,
                    computername);    /* 16 bit */

Net32CharDevQPurgeSelf(pszServername, pszQueueName,
                     computername);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszQueueName (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of a particular printer queue.

computername (const unsigned char LSFAR *) points to an ASCIIZ string specifying the name of a computer whose requests are to be deleted from *pszQueueName*.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_ItemNotFound	2115	The device queue is empty.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_QueueNotFound	2338	A queue does not exist for this request.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg
- DosSemRequest

Related Information

For information about:

- Listing serial device queues on a server, see [Serial Device - NetCharDevQEnum](#).
- Changing the state of a serial device queue, see [Serial Device - NetCharDevQSetInfo](#).

- Deleting the contents of a serial device queue, see [Serial Device - NetCharDevQPurge](#).

NetCharDevQSetInfo or Net32CharDevQSetInfo

NetCharDevQSetInfo or Net32CharDevQSetInfo

The NetCharDevQSetInfo API changes the state of a serial device queue on a server.

The NetCharDevQSetInfo can set the *cq1_priority* and the *cq1_devs* fields in the *chardevQ_info_1* data structure.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <chardev.h>

NetCharDevQSetInfo(pszServername, pszQueueName, sLevel,
                  buf, usBuflen, parmnum);    /* 16 bit */

Net32CharDevQSetInfo(pszServername, pszQueueName, ulLevel,
                    buf, ulBuflen, parmnum); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszQueueName</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of a particular printer queue.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in Serial Device Queue Level 1.
<i>buf</i>	(const unsigned char LSFAR *) points to the data structure described in Serial Device Queue Level 1.
<i>parmnum</i>	(16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a single field of the structure is to be passed. If the value is 0, the entire data structure is passed. Otherwise, <i>parmnum</i> can be set to pass only a single field, defined in CHARDEV.H as follows:

SYMBOLIC CONSTANT	VALUE	COMPONENT
CHARDEVQ_PRIORITY_PARMNUM	2	cq1_priority
CHARDEVQ_DEVICES_PARMNUM	3	cq1_devs

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
-------------------	-------	---------

NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_SEM_TIMEOUT	121	A timeout happened from the semaphore API functions.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UseNotFound	2250	The connection cannot be found.
NERR_BadQueuePriority	2335	The queue priority is not valid.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_QueueNotFound	2338	A queue does not exist for this request.
NERR_BadDevString	2340	This list of devices is not valid.
NERR_BadDev	2341	The requested device is not valid.
NERR_InUseBySpooler	2342	This device is already in use by the spooler.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- [DosDevIOCtl](#)
- [DosFsRamSemRequest](#)
- [DosFsCtl](#)
- [DosGetShrSeg](#)
- [DosSemRequest](#)

Related Information

For information about:

- Deleting the contents of a serial device queue, see [Serial Device - NetCharDevQPurge](#).
- Listing serial device queues on a server, see [Serial Device - NetCharDevQEnum](#).

Server Category

This category includes the following APIs:

[Server - NetGetDCName](#)
[Server - NetServerAdminCommand](#)
[Server - NetServerDiskEnum](#)
[Server - NetServerEnum2](#)
[Server - NetServerGetInfo](#)
[Server - NetServerSetInfo](#)

Server APIs enable most applications to perform remote administration tasks on either a local or remote server. Server APIs are used with the ACCESS.H, SERVER.H, and NETCONS.H header files.

Any user or application assigned Administrator authority on a local or remote server can perform administrative tasks on that server. Those tasks include controlling its operation, user access, and resource sharing. A user can be given administrative privilege with the NetUserAdd and NetUserSetInfo APIs (see [User Category](#)), or the NetShareAdd and NetShareSetInfo APIs (see [Share Category](#)).

Certain low-level parameters that affect the operation of a server can be examined and changed by calling the NetServerGetInfo and NetServerSetInfo APIs. These parameters are defined in the IBMLAN.INI file of the server.

Other changes in the operation of a server require the execution of one of the NET commands, such as NET USE or NET SHARE. To run a NET command on a server, an application calls the NetServerAdminCommand function. This function also accepts any OS/2 command for execution on the server.

To obtain a list of servers available to perform remote administration, an application calls the NetServerEnum2 function. This function enumerates the set of all servers visible on the network. The type of NetServerEnum2 matches the bit mask in the field. To obtain a list of local drives, an application calls the NetServerDiskEnum function.

DOS Considerations

Under DOS, the APIs in the Server category make it possible for remote administrative tasks to be performed on a remote server. NetServerEnum2 can be run on either a local requester or remote server; all of the other Server functions are run on a remote server. Attempting to run NetServerAdminCommand or NetServerGetInfo on a local requester returns NERR_RemoteOnly.

Server Data Structures

To set up a server or to reconfigure an existing server, use *server_info_1*, *server_info_2*, or *server_info_3* with NetServerSetInfo.

NetServerGetInfo returns configuration information at four levels by way of *server_info_0* (server name only), *server_info_1*, *server_info_2*, and *server_info_3*.

The lists of server information returned by NetServerEnum2 are limited to level 0 or level 1.

NetServerAdminCommand uses no data structure.

Server Level 0

```
struct server_info_0 {
    unsigned char sv0_name[CNLEN + 1];
};
```

where *sv0_name* is an ASCIIZ string containing the name of a server.

Server Level 1

At level 1, NetServerEnum2, NetServerGetInfo, and NetServerSetInfo use the *server_info_1* data structure as defined in SERVER.H:

```
struct server_info_1 {
    unsigned char          sv1_name[CNLEN + 1];
    unsigned char          sv1_version_major;
    unsigned char          sv1_version_minor;
    unsigned long          sv1_type;
    unsigned char LSFAR * LSPTR sv1_comment;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *sv1_version_major* is the major release version number of the LAN Server software. The following are the possible values for this field:

SYMBOLIC CONSTANT	VALUE	BITS	MEANING
MAJOR_VERSION_MASK	0x0F	0-3	Major version number
-	0x00	4-7	Unlimited server
-	0x10	4-7	Limited server
-	0x20	4-7	Peer server

- *sv1_version_minor* is the minor release version number of the LAN Server software.
- *sv_x_type* indicates the type of software the computer is running as defined in the header file SERVER.H:

SYMBOLIC CONSTANT	BIT MASK	TYPE OF SOFTWARE
SV_TYPE_WORKSTATION	0x00000001	Requester
SV_TYPE_SERVER	0x00000002	Server

SV_TYPE_SQLSERVER	0x00000004	SQL server
SV_TYPE_DOMAIN_CTRL	0x00000008	Domain controller
SV_TYPE_DOMAIN_BAKCTRL	0x00000010	Backup domain controller
SV_TYPE_TIME_SOURCE	0x00000020	Time server
SV_TYPE_AFP	0x00000040	Apple File Protocol (AFP) service
SV_TYPE_NOVELL	0x00000080	Novell
SV_TYPE_DOMAIN_MEMBER	0x00000100	Domain member
SV_TYPE_PRINTQ_SERVER	0x00000200	Print server
SV_TYPE_DIALIN_SERVER	0x00000400	Dial-in server
SV_TYPE_SERVER_UNIX	0x00000800	UNIX server
SV_TYPE_NT	0x00001000	NT server
SV_TYPE_WFW	0x00002000	Windows for Workgroup server
SV_TYPE_OK_BACKUP	0x00010000	OK to become a backup
SV_TYPE_BACKUP	0x00020000	Backup browse server
SV_TYPE_MASTER	0x00040000	Master browse server
SV_TYPE_BROWSER	0x00070000	All browse servers
SV_TYPE_DOMAIN_MASTER	0x00080000	Domain master server
SV_TYPE_SERVER_OSF	0x00100000	OSF server
SV_TYPE_SERVER_VMS	0x00200000	VMS server
SV_TYPE_SERVER_WINDOWS	0x00400000	Windows server
SSV_TYPE_DCE	0x10000000	DCE Server
SV_TYPE_ALL	0xFFFFFFFF	All types of servers

NOTE: X can be 1, 2, or 3.

Only user and group information is replicated to the backup domain controller.

- *sv1_comment* points to an ASCIIZ string containing a comment describing the server. A null string indicates that there is no comment.

Server Level 2

```

struct server_info_2 {
    unsigned char      sv2_name[CNLEN+1];
    unsigned char      sv2_version_major;
    unsigned char      sv2_version_minor;
    unsigned long       sv2_type;
    unsigned char LSFAR * LSPTR sv2_comment;
    unsigned long       sv2_ulist_mtime;
    unsigned long       sv2_glist_mtime;
    unsigned long       sv2_alist_mtime;
    unsigned short      sv2_users;
    unsigned short      sv2_disc;
    unsigned char LSFAR * LSPTR sv2_alerts;
    unsigned short      sv2_security;
    unsigned short      sv2_auditing;
    unsigned short      sv2_numadmin;
    unsigned short      sv2_lanmask;
    unsigned short      sv2_hidden;
    unsigned short      sv2_announce;
    unsigned short      sv2_anndelta;
    unsigned char       sv2_guestacct[UNLEN + 1];
    unsigned char       sv2_pad1;
    unsigned char LSFAR * LSPTR sv2_userpath;
    unsigned short      sv2_chdevs;
    unsigned short      sv2_chdevq;
    unsigned short      sv2_chdevjobs;
    unsigned short      sv2_connections;
    unsigned short      sv2_shares;
    unsigned short      sv2_openfiles;
    unsigned short      sv2_sessopens;
    unsigned short      sv2_sessvcs;
    unsigned short      sv2_sessreqs;
    unsigned short      sv2_opensearch;
    unsigned short      sv2_activelocks;
    unsigned short      sv2_numreqbuf;
    unsigned short      sv2_sizreqbuf;
    unsigned short      sv2_numbigbuf;
    unsigned short      sv2_numfiletasks;
    unsigned short      sv2_alertsched;
    unsigned short      sv2_erroralert;
    unsigned short      sv2_logonalert;
    unsigned short      sv2_accessalert;
    unsigned short      sv2_diskalert;
    unsigned short      sv2_netioalert;
    unsigned short      sv2_maxauditsz;
    unsigned char LSFAR * LSPTR sv2_srvheuristics;
};

```

where:

- The first five fields in this data structure are identical to those in the previous level. See [Server Level 1](#) for the type of software the computer is running.
- *sv2_ulist_mtime* indicates the last time (in seconds from 00:00:00, 1 January 1970) the users list was changed.
- *sv2_glist_mtime* indicates the last time (in seconds from 00:00:00, 1 January 1970) the groups list was changed.
- *sv2_alist_mtime* indicates the last time (in seconds from 00:00:00 1 January 1970) the access control list was changed.
- *sv2_users* indicates the number of users that are allowed on the server.
- *sv2_disc* indicates the autodisconnect time (in minutes). A session is disconnected if it is idle longer than the time specified by *sv2_disc*. If *sv2_disc* is SV_NODISC, autodisconnect is not enabled.
- *sv2_alerts* points to an ASCII string containing the list of user names on the alert table of the server. Spaces separate the names.
- *sv2_security* specifies the security type of the server. It is set to SV_USERSECURITY, which is defined as follows in SERVER.H:

SYMBOLIC CONSTANT	VALUE	TYPE OF SECURITY
-------------------	-------	------------------

SV_SHARESECURITY	0	Share-level
SV_USERSECURITY	1	User-level

- *sv2_auditing* indicates whether auditing is enabled on the server. If 0, auditing is disabled. If not 0, the server audits LAN Server activities, as described in [Auditing Category](#).
- *sv2_numadmin* indicates the number of administrators a server can accommodate at the same time.
- *sv2_lanmask* determines the order in which the network device drivers are served.
- *sv2_hidden* determines whether the server is visible to other computers in the same domain. *sv2_hidden* is defined as follows in SERVER.H:

SYMBOLIC CONSTANT	VALUE	MEANING
SV_VISIBLE	0	Visible server
SV_HIDDEN	1	Hidden server, not visible

- *sv2_announce* specifies the network announce delta (in seconds), which determines how often the server is to be announced to other computers on the network.
- *sv2_arndelta* specifies the random announce rate (in milliseconds) for *sv2_announce*. The announce interval (*sv2_announce*) can vary by the amount specified in *sv2_arndelta*. For example, it could vary from 9.999 seconds to 10.001 seconds, instead of being exactly 10 seconds each time.
- *sv2_guestacct* is an ASCII string containing the name of a server reserved GUEST user account.
- *sv2_pad1* word-aligns the data structure components.
- *sv2_userpath* points to an ASCII string containing the path name to user directories.
- *sv2_chdevs* indicates the number of serial devices that can be shared on the server.
- *sv2_chdevq* indicates the number of serial device queues that can coexist on the server.
- *sv2_chdevjobs* indicates the number of serial device jobs that can be pending on a server.
- *sv2_connections* indicates the number of connections to netnames that are allowed on a server.
- *sv2_shares* indicates the number of netnames a server can accommodate.
- *sv2_openfiles* indicates the number of files that can be opened at once.
- *sv2_sessopens* indicates the number of files that can be opened in one session.
- *sv2_sessvcs* indicates the maximum number of virtual circuits per client.
- *sv2_sessreqs* indicates the number of simultaneous requests a client can make on any virtual circuit.
- *sv2_opensearch* indicates the number of searches that can be opened at once.
- *sv2_activelocks* indicates the number of file locks that can be active.
- *sv2_numreqbuf* indicates the number of server buffers that are provided.
- *sv2_sizreqbuf* indicates the size (in bytes) of each server buffer.
- *sv2_numbigbuf* indicates the number of 64KB server buffers that are provided.
- *sv2_numfiletasks* indicates the number of processes that can access the operating system at one time.
- *sv2_alertsched* indicates the alert interval (in seconds) for notifying an administrator of a network event.
- *sv2_erroralert* indicates the number of entries that can be written to the error log file during a *sv2_alertsched* interval before notifying an administrator.

- *sv2_logonalert* indicates the number of failed logon attempts to allow a user before notifying an administrator.
- *sv2_accessalert* indicates the number of failed file accesses to allow before issuing an administrative alert.
- *sv2_diskalert* indicates the number of kilobytes of free disk space at which an administrator must be notified that the free space is low.
- *sv2_netioalert* indicates the network I/O error ratio (in tenths of a percent) to allow before notifying an administrator.
- *sv2_maxauditsz* indicates the maximum audit file size (in KB).
- *sv2_srvheuristics* points to an ASCII string of flags used to control the operations of a server.

The heuristics default to values that are optimal for most configurations and normally need not be changed.

For the defaults and ranges of the previous fields, see the IBMLAN.INI parameter descriptions in the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

Server Level 3

Level 3 is not valid for the NetServerEnum2 call.

```

struct server_info_3 {
    unsigned char          sv3_name[CNLEN+1];
    unsigned char          sv3_version_major;
    unsigned char          sv3_version_minor;
    unsigned long          sv3_type;
    unsigned char LSFAR * LSPTR sv3_comment;
    unsigned long          sv3_ulist_mtime;
    unsigned long          sv3_glist_mtime;
    unsigned long          sv3_alist_mtime;
    unsigned short         sv3_users;
    unsigned short         sv3_disc;
    unsigned char LSFAR * LSPTR sv3_alerts;
    unsigned short         sv3_security;
    unsigned short         sv3_auditing;
    unsigned short         sv3_numadmin;
    unsigned short         sv3_lanmask;
    unsigned short         sv3_hidden;
    unsigned short         sv3_announce;
    unsigned short         sv3_anndelta;
    unsigned char          sv3_guestacct[UNLEN + 1];
    unsigned char          sv3_pad1;
    unsigned char LSFAR * LSPTR sv3_userpath;
    unsigned short         sv3_chdevs;
    unsigned short         sv3_chdevq;
    unsigned short         sv3_chdevjobs;
    unsigned short         sv3_connections;
    unsigned short         sv3_shares;
    unsigned short         sv3_openfiles;
    unsigned short         sv3_sessopens;
    unsigned short         sv3_sessvcs;
    unsigned short         sv3_sessreqs;
    unsigned short         sv3_opensearch;
    unsigned short         sv3_activelocks;
    unsigned short         sv3_numreqbuf;
    unsigned short         sv3_sizreqbuf;
    unsigned short         sv3_numbigbuf;
    unsigned short         sv3_numfiletasks;
    unsigned short         sv3_alertsched;
    unsigned short         sv3_erroralert;
    unsigned short         sv3_logonalert;
    unsigned short         sv3_accessalert;
    unsigned short         sv3_diskalert;
    unsigned short         sv3_netioalert;
    unsigned short         sv3_maxauditsz;
    unsigned char LSFAR * LSPTR sv3_srvheuristics;
    unsigned long          sv3_auditedevents;
    unsigned short         sv3_autoprofile;
    unsigned char LSFAR * LSPTR sv3_autopath;
};

```

where:

- The first 44 fields in this data structure are identical to those in the previous level. See the heuristics information under [Server Level 0](#). Note the types of software the computer can run in [Server Level 1](#).
- *sv3_auditedevents* is the audit event control mask. Setting bits in the mask to 1 enables auditing of events according to the following symbolic constants defined in AUDIT.H:

SYMBOLIC CONSTANT	MEANING
SVAUD_SERVICE	Service state change
SVAUD_GOODSESSLOGON	Successful session logon requests
SVAUD_BADSESSLOGON	Unsuccessful session logon requests
SVAUD_SESSLOGON	All session logon and logoff requests
SVAUD_GOODNETLOGON	Successful network logon requests
SVAUD_BADNETLOGON	Unsuccessful network logon requests
SVAUD_NETLOGON	All network logon and logoff requests
SVAUD_LOGON	All logon and logoff requests (network and session)
SVAUD_GOODUSE	Successful share requests
SVAUD_BADUSE	Unsuccessful share requests
SVAUD_USE	All share requests, regardless of GOODUSE or BADUSE switches
SVAUD_USERLIST	Changes to the user or group account database
SVAUD_PERMISSIONS	Changes to the access control list database
SVAUD_RESOURCE	Resource access as defined by the per-resource auditing options specified in the access control list
SVAUD_LOGONLIM	Logon limit violations

- *sv3_autoprofile* is the bit mask for automatic server configuration as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
SW_AUTOPROF_LOAD_MASK	0x1	Autoload at server startup
SW_AUTOPROF_SAVE_MASK	0x2	Autosave at server shutdown

- *sv3_autopath* indicates where to save and load the server information for autoload.

Server Level 4

Level 4 is not valid for the NetServerEnum2 call.

```
struct server_info_4 {
    unsigned char          sv4_name[CNLEN+1];
    unsigned char          sv4_version_major;
    unsigned char          sv4_version_minor;
    unsigned long          sv4_type;
    unsigned char LSFAR * LSPTR sv4_comment;
    unsigned long          sv4_ulist_mtime;
    unsigned long          sv4_glist_mtime;
    unsigned long          sv4_alist_mtime;
    unsigned short         sv4_users;
    unsigned short         sv4_disc;
    unsigned char LSFAR * LSPTR sv4_alerts;
    unsigned short         sv4_security;
    unsigned short         sv4_auditing;
    unsigned short         sv4_numadmin;
    unsigned short         sv4_lanmask;
    unsigned short         sv4_hidden;
    unsigned short         sv4_announce;
    unsigned short         sv4_arndelta;
    unsigned char          sv4_guestacct[UNLEN + 1];
    unsigned char          sv4_pad1;
    unsigned char LSFAR * LSPTR sv4_userpath;
    unsigned short         sv4_chdevs;
    unsigned short         sv4_chdevq;
    unsigned short         sv4_chdevjobs;
    unsigned short         sv4_connections;
    unsigned short         sv4_shares;
    unsigned short         sv4_openfiles;
    unsigned short         sv4_sessopens;
    unsigned short         sv4_sessvcs;
    unsigned short         sv4_sessreqs;
    unsigned short         sv4_opensearch;
    unsigned short         sv4_activelocks;
    unsigned short         sv4_numreqbuf;
    unsigned short         sv4_sizreqbuf;
    unsigned short         sv4_numbigbuf;
    unsigned short         sv4_numfiletasks;
    unsigned short         sv4_alertsched;
    unsigned short         sv4_erroralert;
    unsigned short         sv4_logonalert;
    unsigned short         sv4_accessalert;
    unsigned short         sv4_diskalert;
    unsigned short         sv4_netioalert;
    unsigned short         sv4_maxauditsz;
    unsigned char LSFAR * LSPTR sv4_srvheuristics;
    unsigned long          sv4_auditedevents;
    unsigned short         sv4_autoprofile;
    unsigned char LSFAR * LSPTR sv4_autopath;
    unsigned char LSFAR * LSPTR sv4_cell;
    unsigned char LSFAR * LSPTR sv4_resdom;
};
```

where:

- The first 47 fields in this data structure are identical to those in the previous level.
- *sv4_cell* is an ASCIIZ string containing the name of the cell the server is configured in.
- *sv4_resdom* is an ASCIIZ string containing the name of the resource domain the server belongs to.

Note: This level is only valid for DSS servers.

Server Level 10

```

struct server_info_10 {
    unsigned char          sv10_name[CNLEN + 1];
    unsigned char LSFAR *LSPTR sv10_cell;
    unsigned char LSFAR *LSPTR sv10_resdom;
};

```

where:

- The first field in this data structure is identical to that in the previous level.
- *sv10_cell* is an ASCIIZ string containing the name of the cell the server is configured in.
- *sv10_resdom* is an ASCIIZ string containing the name of the resource domain the server belongs to.

Note: This level is only valid for DSS servers.

Server Level 20

```

struct server_info_20 {
    unsigned long    si20_reserved;
    unsigned short   si20_count;
};

```

where:

- *si20_reserved* must be 0 and is reserved for future use.
- *si20_count* indicates the number of *server_record_id_list* structures that immediately follow the *server_info_20* structure.

Server Record ID List

```

struct server_record_id_list {
    unsigned char LSFAR * LSPTR    sril_srv_rec_ID;
    unsigned char LSFAR * LSPTR    sril_remark;
};

```

where:

- *sril_srv_rec_ID* is an ASCIIZ string containing a server record ID. The server record identifier is retrieved from field 12 of the enabled server record found in the file \BMLAN\RPL\RPL.MAP.
- *sril_remark* points to an ASCIIZ string that contains descriptive information about the server record ID. The remark can be no longer than 80 bytes. The remark field is not optional and does not return a NULL. The server record remark is retrieved from field 7 of the enabled server record found in the file \BMLAN\RPL\RPL.MAP. The server record remark must not contain embedded blank spaces. The tilde (~) character is used as the word delimiter.

Related Information

For information about:

- Domains, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Remote administration of NET commands, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Server heuristics, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetGetDCName or Net32GetDCName

NetGetDCName or Net32GetDCName

Given a domain name, the NetGetDCName API returns the name of the domain controller, if there is one. If you specify the null domain name, the API returns the name of the domain controller (DC) of the primary domain.

Restrictions

This API can be called from DLS and OS/2 workstations. This API does not have any access requirements.

Syntax

```
#include <netcons.h>
#include <access.h>

NetGetDCName(pszServername, pszDomainName, buf,
             usBuflen);           /* 16 bit */

Net32GetDCName(pszServername, pszDomainName, buf,
               ulBuflen);         /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- pszDomainName* (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the domain or resource domain. Also, //resdom syntax is recognized in this field for DSS.
- buf* (unsigned char LSFAR *) points to the buffer for the name of the domain controller to be returned.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.

NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCNotFound	2453	No domain controller was found on this domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosDeleteMailslot
- DosFsCtl
- DosGetShrSeg
- DosMakeMailslot

Remarks

If the return code is 0 (success), the buffer contains an ASCIIZ string representing the name of the domain controller as a UNC name; for example, `\\server`.

Because NetGetDCName attempts to find the domain controller for the specified domain each time it is called, this function can affect the performance of applications that call it often.

NetServerAdminCommand or Net32ServerAdminCommand

NetServerAdminCommand or Net32ServerAdminCommand

The NetServerAdminCommand API runs a command on a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <server.h>

NetServerAdminCommand(pszServername, command, pResult,
                     buf, usBuflen, pusBytesReturned,
                     pusBytesAvail);          /* 16 bit */

Net32ServerAdminCommand(pszServername, command, pResult,
```

```

buf, ulBuflen, pulBytesReturned,
pulBytesAvail);          /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- command* (const unsigned char LSFAR *) points to an ASCIIZ string containing the command to run.
- pResult* (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the returned exit code of the executed command.
- buf* (unsigned char LSFAR *) points to the output of the returned command.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_FILENAME_EXCED_RANGE	206.	The file name is longer than 8 characters or the extension is longer than characters.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ExecFailure	2315	A failure occurred when executing a

		remote administration command.
NERR_TmpFile	2316	A failure occurred when opening a remote temporary file.
NERR_TooMuchData	2317	The data returned from a remote administration command has been truncated to 64KB.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

The NetServerAdminCommand function is a remote form of the C language library *system()* function.

When run remotely, NetServerAdminCommand sets the environment of the server as follows:

- The default drive and directory is C:\IBMLAN\NETPROG.
- The PATH environment is not set (NULL).

When run locally, NetServerAdminCommand sets the environment of the server as follows:

- The current drive and directory of the caller are used.
- The PATH environment is set to the user's default path.

Related Information

For information about:

- Listing available servers, see [Server - NetServerEnum2](#).
- Running a program on a remote server, see [Remote Utility - NetRemoteExec](#).

NetServerDiskEnum or Net32ServerDiskEnum

NetServerDiskEnum or Net32ServerDiskEnum

The NetServerDiskEnum API retrieves a list of disk drives on a workstation.

The drive names in the list are consecutive strings, each containing a drive letter, a colon (:), and a null string terminator (\0). For example, the following can be returned for a server having two diskette drives (A and B), one hard-disk drive (C), and one RAM drive (E):

```
A:\0B:\0C:\0E:\0
```

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority or server operator privilege are required.

Syntax

```
#include <netcons.h>
```

```
#include <server.h>

NetServerDiskEnum(pszServername, sLevel,
                  buf, usBuflen, pusEntriesReturned,
                  pusEntriesAvail);          /* 16 bit */

Net32ServerDiskEnum(pszServername, ulLevel,
                   buf, ulBuflen, pulEntriesReturned,
                   pulEntriesAvail);        /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0.

buf (unsigned char LSFAR *) points to the returned list of disk drive names.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Listing the shared resources of a server, see [Server - NetServerEnum2](#).
- Listing available servers, see [Server - NetServerEnum2](#).

NetServerEnum2 or Net32ServerEnum2

NetServerEnum2 or Net32ServerEnum2

The NetServerEnum2 API enumerates the set of all servers visible on the network. The type of NetServerEnum2 matches the bit mask in the field.

The NetServerEnum2 API replaces the NetServerEnum API, which is now obsolete.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can only issue this call to a remote LAN Server workstation. This API does not have any access authority requirements.

Directory and Security Server Only

The NetServerEnum2 API now recognizes the //resdom syntax in the domain name field. When the return code is NERR_DCEError, the buffer will hold the text of the DCE error.

Syntax

```
#include <netcons.h>
#include <server.h>

NetServerEnum2(pszServername, sLevel, buf,
               usBuflen, pusEntriesReturned,
               pusEntriesAvail, servertype,
               pszDomainName);          /* 16 bit */

Net32ServerEnum2(pszServername, ulLevel, buf,
                 ulBuflen, pulEntriesReturned,
                 pulEntriesAvail, servertype,
                 pszDomainName);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) specifies the level of detail (0 or 1) for the *server_info* data structure, as described in [Server Level 1](#) and [Server Level 2](#). Levels 2 and 3 are not valid for NetServerEnum2.

servertype (unsigned long) specifies the types of servers to enumerate. This parameter is tested against the *sv_x_type* element of each entry. Entries that match at least one of the specified bits are included both in the returned buffer and in the counts returned by *pusEntriesReturned* and *pusEntriesAvail*. The *sv_x_type* element, where *x* is 1, 2, or 3, and its values are defined as follows in SERVER.H:

SYMBOLIC CONSTANT	BIT MASK	TYPE OF SOFTWARE
SV_TYPE_WORKSTATION	0x00000001	Workstation
SV_TYPE_SERVER	0x00000002	Server
SV_TYPE_SQLSERVER	0x00000004	SQL server

SV_TYPE_DOMAIN_CTRL	0x00000008	Domain controller
SV_TYPE_DOMAIN_BAKCTR	0x00000010	Backup domain control
SV_TYPE_TIME_SOURCE	0x00000020	Time server
SV_TYPE_AFP	0x00000040	Apple File Protocol (A
SV_TYPE_NOVELL	0x00000080	
SV_TYPE_NOVELL	0x00000080	Novell
SV_TYPE_DOMAIN_MEMBER	0x00000100	Domain member
SV_TYPE_PRINTQ_SERVER	0x00000200	Print server
SV_TYPE_DIALIN_SERVER	0x00000400	Dial-in server
SV_TYPE_SERVER_UNIX	0x00000800	UNIX server
SV_TYPE_NT	0x00001000	NT server
SV_TYPE_WFW	0x00002000	Windows for Workgroup
SV_TYPE_OK_BACKUP	0x00010000	OK to become a backup
SV_TYPE_BACKUP	0x00020000	Backup browse server
SV_TYPE_MASTER	0x00040000	Master browse server
SV_TYPE_BROWSER	0x00070000	All browse servers
SV_TYPE_DOMAIN_MASTER	0x00080000	Domain master server
SV_TYPE_SERVER_OSF	0x00100000	OSF server
SV_TYPE_SERVER_VMS	0x00200000	VMS server
SV_TYPE_SERVER_WINDOW	0x00400000	Windows server
SSV_TYPE_DCE	0x10000000	DCE Server
SV_TYPE_ALL	0xFFFFFFFF	All types of servers

pszDomainName (unsigned char LSFAR *) points to an ASCIIZ string containing the domain name of the servers to monitor.

A NULL pointer for *pszDomainName* or a pointer to a null value specifies all of the domains that the requester is monitoring.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.

ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BrowserTableIncomplete	2319	The server table was initialized incorrectly.
NERR_NotLocalDomain	2320	This domain is not active on this computer.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned by the DosFsCtl function.

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

The NetServerEnum2 API can obtain only level 0 and level 1 data structures.

This API returns the list of servers in domains the requester is monitoring. Because of its efficiency, NetServerEnum2 is the preferred API to retrieve server names.

The requester maintains a list of servers started in the default domain, the logon domain, and any other domains the requester is configured to monitor. The default domain is specified by the **DOMAIN** field in the Requester section of the IBMLAN.INI file or by the */DOMAIN* parameter on NET START REQUESTER.

The logon domain can be different from the default domain if the user selects a different domain at logon time. The requester can be configured to monitor up to four other domains by specifying these domains on the **OTHDOMAINS** parameter line in the IBMLAN.INI file or in the */OTHDOMAINS* parameter on the NET START REQUESTER command.

The restrictions on the value of the **DOMAIN** parameter apply to the computer on which the NetServerEnum2 API actually is run ; that is, the local computer if the *pszServername* argument is NULL, or the named server if the *pszServername* argument is not NULL.

NetServerGetInfo or Net32ServerGetInfo

NetServerGetInfo or Net32ServerGetInfo

The NetServerGetInfo API retrieves information about a particular server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. User authority is allowed limited access to this API. Administrator authority is required for full access.

Syntax

```
#include <netcons.h>
#include <server.h>

NetServerGetInfo(pszServername, sLevel,
                buf, usBuflen,
                pusBytesAvail);    /* 16 bit */

Net32ServerGetInfo(pszServername, ulLevel,
                  buf, ulBuflen,
                  pulBytesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, 2, 3, 4, 10 or 20, specifying which level of data structure to use, as described in [Server Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been

started.

NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Depending on the level of information requested (by way of the level parameter), the NetServerGetInfo function returns information ranging from the name of the server to a description of the server heuristics, which control the way the server operates.

Level 0 and level 1 information can be accessed remotely by users with user authority.

A valid RPL.MAP will contain at least one enabled server record.

Related Information

For information about:

- Configuring a server, see [Server - NetServerSetInfo](#).
- Server heuristics, see the *LAN Server Network Administrator Reference Volume 2: Performance Tuning*.

NetServerSetInfo or Net32ServerSetInfo

NetServerSetInfo or Net32ServerSetInfo

The NetServerSetInfo API sets operating parameters for a server (either individually or collectively).

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <server.h>

NetServerSetInfo(pszServername, sLevel, buf,
                 usBuflen, parmnum);    /* 16 bit */

Net32ServerSetInfo(pszServername, ulLevel, buf,
                  ulBuflen, parmnum);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) specifies the level of detail (1, 2, or 3) provided by the *server_info* data structure.

buf (const unsigned char LSFAR *) points to the data structure, if *parmnum* is 0. Otherwise, this parameter points to the specific data component that is to be changed.

parmnum (16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a specific field in the data structure is to be passed. If the value is 0, the entire data structure is sent. Otherwise, *parmnum* specifies which field in the data structure is to be sent. If not 0, *parmnum* can be any one of the following:

SYMBOLIC CONSTANT	VALUE	COMPONENT
SV_COMMENT_PARMNUM	5	svX_comment or sv1_comme
SV_DISC_PARMNUM	10	svX_disc
SV_ALERTS_PARMNUM	11	svX_alerts
SV_HIDDEN_PARMNUM	16	svX_hidden
SV_ANNOUNCE_PARMNUM	17	svX_announce
SV_ANNDELTA_PARMNUM	18	svX_anndelta
SV_ALERTSCHED_PARMNUM	37	svX_alertsched
SV_ERRORALERT_PARMNUM	38	svX_erroralert
SV_LOGONALERT_PARMNUM	39	svX_logonalert
SV_ACCESSALERT_PARMNUM	40	svX_accessalert
SV_DISKALERT_PARMNUM	41	svX_diskalert
SV_NETIOALERT_PARMNUM	42	svX_netioalert
SV_MAXAUDITSZ_PARMNUM	43	svX_maxauditsz
NOTE: X = 2 or 3.		

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsRamSemClear
- DosFreeSeg
- DosFsCtl
- DosGetShrSeg
- DosSemClear

Related Information

For information about retrieving the configuration of a server, see [Server - NetServerGetInfo](#).

Services Category

This category includes the following APIs:

[Services - NetServiceControl](#)
[Services - NetServiceEnum](#)

[Services - NetServiceGetInfo](#)
[Services - NetServiceInstall](#)
[Services - NetServiceStatus](#)

LAN Server provides many network services, as well as APIs, you can use to create other services. A service is a program, of any size and function, that other applications can use to perform some set of tasks on the network.

The two most important services provided by LAN Server are Requester and Server, which provide the majority of the software required to operate a local area network.

The service programs are installed in the \IBMLAN\SERVICES directory. LAN Server updates the [services] section of the IBMLAN.INI file each time LAN Server is started, adding any new services, their configuration parameters, and their default values.

In the IBMLAN.INI file, you can control when a service is started by changing one of the following two parameters:

[Requester] section, WRKSERVICES parameter

Specifies that a service is to be started at the same time as the Requester service. For example:

```
WRKSERVICES=MESSENGER
```

[Server] section, SRVSERVICES parameter

Specifies that a service is to be started at the same time as the Server service. For example:

```
SRVSERVICES = NETLOGON,LSSERVER,ALERter,REPLICATOR
```

If a service is not started when LAN Server is started, you can start it either from the Network Services window of the LAN Requester graphical user interface (see *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*) or from an OS/2 command prompt with the NET START command. See *Directory and Security Server Commands and Utilities* for more information about the NET START command and the parameters available for each service.

[Services Provided with LAN Server](#) describes the services shipped with LAN Server. [Writing a Service](#) discusses how to create a new service.

DOS Considerations

Under DOS, all the services can be paused and continued with the NetServiceControl API, and all but the redirector can be stopped.

Services Provided with LAN Server

With user authority, your application can call all of the following services locally. With administrative authority, your application can call them remotely, as well.

Alerter

Generates an alert notification whenever a system event occurs, determines which requesters are registered to receive notifications about events in the same class, and sends the notification to those requesters.

The Alerter service registers itself to receive all print, error-log, and administrative alerts. It creates and registers a mail slot to receive alerts triggered by these events. When it receives an alert, the Alerter service converts the information to text and sends an event notification by calling the NetMessageBufferSend API.

The Alerter service sends alert notifications about an individual print job only to the user who submitted the job. It sends administrative and error-log alert notifications to all users listed in the *sv2_alerts* field of the *server_info_2* data structure. A list of these users can be retrieved by calling the NetServerGetInfo API.

You can set up thresholds for various server parameters in the IBMLAN.INI file to determine when an alert is generated. You also can specify in the IBMLAN.INI file where the alerts should be sent.

To interface your application with this service, use the APIs in the Service and Alert Categories.

For more information about alert functions, see [Alert Category](#).

DCDB Replicator	Replicates the \IBMLAN\DCDB subdirectory (except remote IPL images) from a domain controller to other servers defined as backup domain controllers. When used with the Netlogon service, the DCDB Replicator service ensures that users always can log on to a domain, even if the domain controller fails or is busy. The DCDB Replicator service is separate from the Replicator service, and both can reside and function independently of each other on the domain controller or backup domain controllers.
Generic Alerter (Genalert)	Enables the server to build and send Systems Network Architecture (SNA) alerts. The Generic Alerter service is notified by the Alerter service when certain LAN alerts occur. The Generic Alerter service then uses the FFST/2 program to build software generic alerts that conform to the SNA format and sends them either to the NetView program or to IBM LAN Network Manager using the FFST/2 router. The Generic Alerter service can be started only if the FFST/2 program is running.
LSserver	Provides DOS LAN Services support and logical server functions. The logical server supports remote requests from requesters for activities such as spooling, querying users, logon, and logoff. The LSserver service is started and stopped by the Server service. Increase the default value of 3 for the <i>srvpipes</i> parameter in the IBMLAN.INI file only if the administrator is detecting unusually long logon times, printing responses, and so on.
Messenger	<p>Enables users to (1) send and receive messages, (2) save messages in the log file, (3) add and delete messaging names, and (4) forward messages to another requester. The Messenger service does all these things by calling the Message category functions (see Message Category). The Messenger service receives messages sent from remote workstations by way of the NetMessageBufferSend or NetMessageFileSend APIs. For more details, see the <i>LAN Server Network Administrator Reference Volume 2: Performance Tuning</i>.</p> <p>To interface your application with this service, use the APIs in the Service and Messaging Categories.</p>
Netlogon	Allows servers in a domain to receive updated copies of the user and group definitions file (NET.ACC), which is maintained on the domain controller. Enables domain controllers and additional servers that function as a domain controller backup to satisfy logon requests. Controls (1) the length of time a user can be logged on to the network, (2) the password expiration time, and (3) number of user passwords that must be unique. The Server service must be started before the Netlogon service can be started.
Netrun	Supports the remote execution of programs. If the Netrun service is paused, requests to run a program remotely are denied. When the Netrun service is removed, each process started by the server is ended by the OS/2 DosKillProc API. The Server service must be started before the Netrun service can be started.
Peer	<p>Gives a requester (called a peer server) some of the capabilities of a server, allowing the owner to share directories, named pipes, one printer queue, and one communication device queue with other users on the network. It lets one user at a time connect to the peer server.</p> <p>Peer workstations are not started from the domain. They are administered by the owner of the peer workstation, rather than the network administrator, who has no control over the resources at a peer workstation.</p> <p>Because peer workstations are not part of the domain, they can be administered only from the command line and do not support alias names.</p>
Remote Initial Program Load (Remoteboot)	<p>Enables the Server service to support remote IPL of DOS and OS/2 requesters, including requesters without local media (CD-ROM, hard disk, or diskette drives).</p> <p>To interface your application with this service, use the APIs in the Service and RIPL Categories.</p>
Replicator	Provides file replication service. It copies files from a master location on a server to one or more servers or requesters requiring a copy of the data so that an information base can be maintained simultaneously on two or more workstations. The server sending the data is called the <i>exporter</i> . The requesters or servers receiving the data are called <i>importers</i> . This service cannot be paused.
Requester	<p>Maintains most internal information and activates the network device drivers. If the Requester service is paused, no redirection can be established to files, printers, or serial devices. The Requester service enables a workstation to use local area network resources and services. The Requester service must be running for any other LAN Server service to run, and it cannot be removed while the Server service is started.</p> <p>To interface your application with this service, use the APIs in the Service and Requester Categories.</p>
Server	Provides the basic function needed to share local resources over the network. If the Server service is

paused, all further requests for resources are denied. However, all current uses of shared resources continue.

Users with administrative authority are allowed to connect to the server even when it is paused. This allows the server to be managed remotely by an administrator. To interface your application with this service, use the APIs in the Server and Service Categories.

Timesource

Designates a server with a reliable time and date with which other workstations in the same domain or other domains can synchronize. Timesource does not keep time. It only provides the means for other workstations on the network to identify a server with a reliable clock. The keeping of a reliable clock is external to this service.

An additional server can be designated a Timesource server by starting the Timesource service from the command line or by updating IBMLAN.INI to add Timesource to the SRVSERVICES statement. Multiple Timesource servers can exist on a network; it is the responsibility of the requesting application to determine with which Timesource server it will synchronize. OS/2 LAN Requester and LAN Server do not use the Timesource support; they always synchronize with the domain controller.

Uninterruptible Power Supply (UPS)

Provides protection against loss of data during power failures. When a power interruption occurs at a server, users who have active sessions with the affected server are notified of an impending shutdown. They have the opportunity to save and close open files on their workstation before the server shuts down. The UPS service then stops all services and writes all data in cache memory to permanent storage. The service cannot be paused or continued.

Servers must have an uninterruptible power supply, such as a battery, connected to a dedicated serial port.

Additional Services Provided with DSS

DSSDCE

Starts and stops DCE services with the requester.

DIRSYNC

Synchronizes DSS entries in the directory server namespace to the domain controller's DCDB. This allows non-DSS clients and servers to interoperate with DSS.

If the resource domain contains DSS servers only and access to resources by non-DSS clients is not required, then the DIRSYNC should be disabled.

LSPWSYNC

Synchronizes password changes between DCE and LAN Server.

Writing a Service

This section includes the following APIs:

- NetServiceControl
- NetServiceEnum
- NetServiceGetInfo
- NetServiceInstall
- NetServiceStatus

This section discusses considerations for writing a service. The service APIs start and control service programs. These APIs are used with the SERVICE.H and NETCONS.H header files.

With only user authority, your application can call all of the service functions locally. With administrative authority, your application can call them remotely, as well.

Requirements

When designing a service to use with the LAN Server software, keep in mind the following requirements:

- An executable file of a service must be listed under the Services section in the IBMLAN.INI file.
- A service must not call screen or keyboard functions; this can be done indirectly by calling a pop-up function such as the VioPopup API (an OS/2 program function).
- A service must, by calling the NetServiceStatus API, dynamically notify the LAN Server software about a change in the service's status, so that other applications can respond correctly to the change.
- When any signal is sent by an application, a service must respond by calling the NetServiceControl API to change its current state of operation.

An IBMLAN.INI entry of a service must include the name of the service and a valid path name of an executable file used to start the service; the service entry can have additional parameters that supply other information. Applications requesting to use the service can call the functions in the Configuration category to obtain the additional information. For information about using the Configuration functions and a description of IBMLAN.INI components, see [Configuration Category](#).

Differences in 16-Bit and 32-bit Services

The only differences between a 32-bit service and a 16-bit service are details related to signal handling.

16-bit Services

In 16-bit services, you register a signal handler function that accepts the SIG_PFLG_A signal and acts on the information. To register the signal handler, call DosSetSigHandler() in your main program:

```
#define INCL_DOSSIGNALS
#include <os2.h>          /* From the TOOLKT13\C\INCLUDE directory */
#include <netcons.h>      /* From the header files supplied with LAN Server */
#include <service.h>      /* From the header files supplied with LAN Server */
.
.
.
int main (void) /* or include argc, etc., to taste */
{
.
.
.
    DosSetSigHandler((PFNSIGHANDLER)&sig_handler, /* Signal handler address */
                     NULL,                        /* Ignore previous handler */
                     0,                          /* Ignore previous action */
                     SIGA_ACCEPT,                /* Process the signal */
                     SERVICE_RCV_SIG_FLAG))      /* This constant is the
                                                    same as SIG_PFLG_A */
.
.
.
}
```

The outline of the signal handler looks like this:

```
void sig_handler(USHORT sig_info, USHORT sig_num)
{
    unsigned char opcode, arg;
    struct service_status status;
```



```
}
```

Starting a Service

After a service is started on a computer, the service must:

1. Verify any command-line parameters passed to the service by the calling process.
2. Register a signal handler to interpret opcodes passed by requesting applications. For more information, see [Differences in 16-Bit and 32-bit Services](#).
3. Set its state to `SERVICE_INSTALL_PENDING`, by calling the `NetServiceStatus` API, to notify any requesting applications that it is not ready for use.
4. Complete any other initialization procedures previously defined by the service.
5. Notify the LAN Server software that installation is complete by calling the `NetServiceStatus` API and setting its state to `INSTALLED`.

The `NetServiceInstall` API processes the executable file specified in the `IBMLAN.INI` component of the service by calling the `OS/2 DosExecPgm` API and passing the string of parameters made up of `IBMLAN.INI` parameters and information passed to the *cmdargs* parameter of the `NetServiceInstall` API.

The `DosExecPgm` API runs the service in detached mode, preventing handles from being passed to child processes and preventing screen- and keyboard-oriented calls except through pop-up functions. A service inherits the environment of the parent process, the `NetServiceInstall` API.

If a service includes more than one process, the process that the `DosExecPgm` API initially runs (referred to as the main service process) is the only process that receives standard signals from the `NetServiceControl` API. The main service process is the only one that can issue calls to the `NetServiceStatus` API. A service can transfer the responsibilities of the main service process to another process by setting the `PID` component of the *service_status* data structure (passed to the `NetServiceStatus` API) to the process identification number (`PID`) of the main service process candidate.

After receiving control from the `DosExecPgm` API, a service validates the parameters passed from the `IBMLAN.INI` file. If the parameters are not valid, a service notifies the LAN Server software by calling the `NetServiceStatus` API and ending execution. Otherwise, the service installation continues.

After verifying parameters, a service must start a signal handler that communicates with the `NetServiceControl` API. Using the signal handler, a service specifies its current state (such as `INSTALL_PENDING` or `INSTALLED`) so that the LAN Server software can enable other applications to use the service properly. The signal handler must register a function for the `FlagA` signal, the `SERVICE_REC_SIG_FLAG` opcode, by calling the `DosSetSigHandler` API.

After a service specifies its current state as `INSTALLED`, another application can change or query the state of the service by calling the `NetServiceControl` API. With this function, an application can pass an opcode specifying one of four actions to take. The actions are defined in the `SERVICE.H` header file, as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
<code>SERVICE_CTRL_INTERROGATE</code>	0	Request for general information.
<code>SERVICE_CTRL_PAUSE</code>	1	Pause the service.
<code>SERVICE_CTRL_CONTINUE</code>	2	Continue a paused service.
<code>SERVICE_CTRL_UNINSTALL</code>	3	Shut down a service.

A service can define its own set of valid opcodes. Opcodes that are not valid should default to another opcode such as `SERVICE_CTRL_INTERROGATE`. The LAN Server software defines the following limits:

- A service that does not accept the `SERVICE_UNINSTALL` opcode cannot be removed at any time.
- While in the `SERVICE_INSTALL_PENDING` state, a service can receive only the `SERVICE_UNINSTALL` opcode.

In the following pseudocode, an application and a service communicate by means of a signal handler and the LAN Server software:

Application	LAN Server	Service
Application calls NetServiceControl, specifying an opcode to perform a particular task.	Opcode is sent to the specified service.	Signal handler interprets opcode.
		Performs the task defined by the opcode.
		Calls NetServiceStatus to clear the control semaphore and, if required, change the state of the service.
	Updates service information table according to current state of the service.	
NetServiceControl returns with the appropriate information about the service's current state.		

If performing the task takes a long time (more than a few seconds), the service should make intermediate calls to the NetServiceStatus API.

Stopping a Service

When a service is no longer needed, either the application using it or the service itself should call the NetServiceControl API and specify the appropriate shutdown opcode. After receiving a shutdown opcode, a service must again call the NetServiceStatus API to declare SERVICE_UNINSTALL_PENDING status and then perform any other necessary tasks, such as closing open resources. The final step is to call the OS/2 DosExit API. Immediately before doing this, the service again must change its status, this time to SERVICE_UNINSTALLED.

For an application to disable a service from processing any further requests, the service:

1. Calls the NetServiceStatus API to set a SERVICE_UNINSTALL_PENDING state

2. Runs a cleanup routine, closing any open resources
3. Notifies the LAN Server software that it has been removed by calling the NetServiceStatus API and setting a SERVICE_UNINSTALLED state
4. Ends program execution

To obtain information about services started, an application calls the NetServiceEnum API. To query the state of a service, retrieving its status and code information, an application calls the NetServiceStatus API.

For information about the default LAN Server services and the functions used to control services, see [Services Provided with LAN Server](#).

Time Hints for Starting and Stopping a Service

For the services that take a long time to start or to stop, a mechanism is provided to feed back or hint to the process that started or stopped them. This mechanism allows the process to determine dynamically how long to wait for the service to (1) complete the startup or stop and (2) provide the estimated time to the user, if determined by the application. The service can set the text field *svc2_text* as part of its status information.

The services and programs that start or stop the services can use this mechanism to communicate during the start and stop operations. The control completion pending (CCP) code values are used when a service expects to take a long time to start or stop. Any service that has a nontrivial initialization or shutdown task can use this method.

If a service is not using CCP codes, the *svcs_code* field in the data structure should be set to 0 while the service status is INSTALL_PENDING or UNINSTALL_PENDING.

If a service is using CCP codes, then the *svcs_code* field is mapped as follows:

```
33322222 222221111 111111
21098765 432109876 54321098 76543210

xxxxxxxx  xxxxxxxxH  TTTTTTTT  CCCCCCCC

x = not used (must be 0)
H = hint is given          SERVICE_CCP_QUERY_HINT
T = time to wait           SERVICE_CCP_WAIT_TIME
C = checkpoint number      SERVICE_CCP_CHKPT_NUM
```

Bits that are not used must be set to 0.

The time to wait is the expected time to complete the current operation (start or stop), in tenths of a second.

The checkpoint number is a number that should be incremented, or at least changed to a higher value, each time the service calls the NetServiceStatus API. A service should call the NetServiceStatus API fairly often to keep updating this number, remembering that it is an 8-bit quantity.

A controlling application assumes the service is still active if this checkpoint value constantly changes. For that reason, the code that calls the NetServiceStatus API to update the CCP code should be in the main code path, rather than in some time-triggered thread that might continue even while the current operation has halted as the result of an error.

The "hint" bit informs the application controlling the service that the other (time and count) information is valid. If this bit is set, the field *svc2_text* can contain an ASCIIZ string that, when displayed to a user, provides some information about the current state of the service. The service can continue to use a null string if it does not need to provide text. It is up to the controlling application to make use of the text field, but it can assume that the text is suitable for display to the user.

Services Data Structures

The functions in the Service category use two types of data structures: one type containing basic status information, and the other containing different levels of detail.

The NetServiceControl and NetServiceInstall APIs use the *service_info_2* data structure. The NetServiceEnum and NetServiceGetInfo APIs use the *service_info* data structures (level 0, 1, or 2). The NetServiceEnum API returns information at three levels of detail (0, 1, or 2); the level parameter controls the level of information returned.

The NetServiceStatus API uses the *service_status* data structure.

Service Information Data Structures

Service information data structures are as follows.

Service Level 0

```
struct service_info_0 {
    unsigned char svci0_name[SNLEN+1];
};
```

where *svci0_name* is an ASCIIZ string containing the name of the network service to monitor. The constant SNLEN is defined in the NETCONS.H header file.

Service Level 1

```
struct service_info_1 {
    unsigned char   svcil_name[SNLEN+1];
    unsigned short  svcil_status;
    unsigned long   svcil_code;
    unsigned short  svcil_pid;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *svci1_status* is a bit map denoting the status of the network service. These bits are mapped the same as those for *svcs_code*.
- If the service fails, *svci1_code* is the error code returned. In the case of a pending service, however, this parameter serves as a bit-map mask providing information about that pending service. The values for this field are the same as those for *svcs_code*.
- *svci1_pid* is the process identification number (PID) of a service.

Service Level 2

```
struct service_info_2 {
    unsigned char    svci2_name[SNLEN+1];
    unsigned short   svci2_status;
    unsigned long    svci2_code;
    unsigned short   svci2_pid;
    unsigned char    svci2_text[STXTLEN+1];
};
```

where:

- The first four fields in this data structure are identical to those in the previous level.
- *svci2_text* is a null reserved ASCIIZ string, except for stopped services (SERVICE_UNINSTALLED). In this case, *svci2_text* specifies a related parameter string for the *svci2_code* component. The value in this field cannot be longer than STXTLEN+1 bytes.

Service Status Data Structure

The NetServiceStatus API uses the following data structure:

```
struct service_status {
    unsigned short svcs_status;
    unsigned long  svcs_code;
    unsigned short svcs_pid;
    unsigned char  svcs_text[STXTLEN+1];
};
```

where:

- *svcs_status* specifies the status of the service. This field contains a bit-map mask, defined in the SERVICE.H header file as follows:

BITS	SYMBOLIC CONSTANT	BIT MASK	DESCRIPTION
0-1	SERVICE_INSTALL_STATE	0x03	General status of the service.
2-3	SERVICE_PAUSE_STATE	0x0C	Paused/active status.
4	-	-	Service can or cannot be installed.
5	-	-	Service can or cannot be paused.
6-7	-	-	Reserved.
8-10	SERVICE_REDIR_PAUSED	0x0700	Redirection paused/active.
11-15	-	-	Reserved.

Bits 0 and 1 indicate the general status of the service.

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_UNINSTALLED	0x00	Service stopped.
SERVICE_INSTALL_PENDING	0x01	Service start pending.
SERVICE_UNINSTALL_PENDING	0x02	Service stop pending.

SERVICE_INSTALLED	0x03	Service started.
-------------------	------	------------------

The service currently is in one of the following pause states, and bits 2-3 have these values:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_ACTIVE	0x00	Service active.
SERVICE_CONTINUE_PENDING	0x04	Service continue pending.
SERVICE_PAUSE_PENDING	0x08	Service pause pending.
SERVICE_PAUSED	0x0C	Service paused.

Bit 4 indicates whether the service can be stopped by an application, shown in the following values:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_NOT_UNINSTALLABLE	0x00	Service cannot be stopped.
SERVICE_UNINSTALLABLE	0x10	Service can be stopped.

Bit 5 indicates whether the service can be paused by an application, shown in the following values:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_NOT_PAUSABLE	0x00	Service cannot be paused.
SERVICE_PAUSABLE	0x20	Service can be paused.

Bits 6 and 7 are reserved, with a value of 0.

Bits 8-10 indicate whether particular tasks within the Requester service have been paused, shown by the following values:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_REDIR_DISK_PAUSED	0x100	Redirector for disks paused.
SERVICE_REDIR_PRINT_PAUSED	0x200	Redirector for spooled devices paused.
SERVICE_REDIR_COMM_PAUSED	0x400	Redirector for serial devices paused.
SERVICE_REDIR_PAUSED	0x700	Redirector paused/active.

- *svcs_code* is the error code returned if the designated service stops or fails to start properly. (The exception is a pending service start or stop, where this parameter serves as a bit-map mask providing information about that pending service. A description of this map is included in the pending service start or stop information.) For its primary return codes, the SERVICE.H header file defines high-word values of *svcs_code* as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_UIC_NORMAL	0	Normal.
SERVICE_UIC_BADPARMVAL	3051	Incorrect parameter value specified.
SERVICE_UIC_MISSPARM	3052	Missing parameter.
SERVICE_UIC_UNKPARM	3053	Unknown parameter specified.

SERVICE_UIC_RESOURCE	3054	Insufficient resource.
SERVICE_UIC_CONFIG	3055	Configuration faulty.
SERVICE_UIC_SYSTEM	3056	OS/2 program error.
SERVICE_UIC_INTERNAL	3057	Internal error encountered.
SERVICE_UIC_ambiguousparm	3058	Ambiguous parameter name.
SERVICE_UIC_dupparm	3059	Duplicated parameter.
SERVICE_UIC_KILL	3060	Ended by the NetServiceControl API.
SERVICE_UIC_EXEC	3061	Could not run service program file.
SERVICE_UIC_SUBSERV	3062	Subservice did not start.
SERVICE_UIC_CONFLPARM	3063	Conflict in the value or use of these parameters.
SERVICE_UIC_BADCOMPNAME	3064	Not a valid computer name.

The SERVICE.H header file defines low-word values for *svcs_code* as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_UIC_M_NULL	0	Normal.
SERVICE_UIC_M_MEMORY	3070	Insufficient memory.
SERVICE_UIC_M_DISK	3071	Insufficient disk space.
SERVICE_UIC_M_THREADS	3072	Unable to create thread.
SERVICE_UIC_M_PROCESSES	3073	Unable to create process.
SERVICE_UIC_M_SECURITY	3074	Security failure.
SERVICE_UIC_M_LANROOT	3075	Incorrect or missing default path.
SERVICE_UIC_M_REDIR	3076	Network software not started.
SERVICE_UIC_M_SERVER	3077	Server software not started.
SERVICE_UIC_M_SEC_FILE_ERR	3078	Server could not access UAS database.
SERVICE_UIC_M_FILES	3079	Not supported.
SERVICE_UIC_M_LOGS	3080	\IBMLAN\LOGS directory not valid.
SERVICE_UIC_M_LANGROUP	3081	Domain specified could not be used.
SERVICE_UIC_M_MSGNAME	3082	Computer name is being used as a message name on another computer.
SERVICE_UIC_M_ANNOUNCE	3083	Requester did not announce the server name.
SERVICE_UIC_M_UAS	3084	The UAS database is not configured correctly.

When the high-word value is SERVICE_UIC_CONFIG (numeric value of 3055), the low-word value can be one of the following:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_UIC_M_SERVER_SEC_ERR	3085	Server is not running with user-level security.
SERVICE_UIC_M_WKSTA	3087	The requester detected a problem with the cable connection.
SERVICE_UIC_M_ERRLOG	3088	View your error log (NET.ERR) for details.
SERVICE_UIC_M_LAZY	3091	The 386 server cannot be started because CACHE386.EXE is not running.

In the case of a pending service start or stop (SERVICE_INSTALL_PENDING or SERVICE_UNINSTALL_PENDING for the *svcs_status* field), *svcs_code* serves as a bit-map mask providing information about that pending service. The SERVICE.H header file defines this bit mask for *svcs_code* as follows:

BITS	START PENDING CODE	BIT MASK	MEANING
0-7	SERVICE_CCP_NO_HINT SERVICE_IP_NO_HINT	0x0	No reason given for start pending.
0-7	SERVICE_CCP_CHKPT_NUM SERVICE_IP_CHKPT_NUM	0xFF	Checkpoint number incremented each time the service calls the NetServiceStatus API (Installer assumes incrementing denotes a valid service.)
8-15	SERVICE_CCP_WAIT_TIME SERVICE_IP_WAIT_TIME	0xFF00	Time to wait; expected time (tenths of a second) to start install or to stop install.
16	SERVICE_CCP_QUERY_HINT SERVICE_IP_QUERY_HINT	0x1000	Reason given for start or stop pending.
17-31	-	-	Reserved; must be 0.

- *svcs_pid* is the process identification number (PID) of a service.
- *svcs_text* is a null reserved ASCII string, unless the service specified by *svcs_pid* is stopped. In this case, *svcs_text* must specify a parameter string related to the *svcs_code* component.

NetServiceControl or Net32ServiceControl

NetServiceControl or Net32ServiceControl

The NetServiceControl API manages the operations of network services.

Restrictions

This API can be called from DLS and OS/2 workstations. User authority is allowed limited access to this API.

Administrator authority is required for full access.

Syntax

```
#include <netcons.h>
#include <service.h>

NetServiceControl(pszServername, service, opcode,
                  arg, buf, usBuflen);          /*16 bit*/

Net32ServiceControl(pszServername, service, opcode,
                   arg, buf, ulBuflen);        /*32 bit*/
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

service (const unsigned char *) points to an ASCIIZ string containing the name of the network service being managed.

opcode (16-bit unsigned char or unsigned short) indicates the action to perform on the service, defined in the SERVICE.H header file as follows:

SYMBOLIC CONSTANT/BITS	VALUE	MEANING
SERVICE_CTRL_INTERROGATE	0	Interrogate service status
SERVICE_CTRL_PAUSE	1	Pause service.
SERVICE_CTRL_CONTINUE	2	Continue service.
SERVICE_CTRL_UNINSTALL	3	Stop service.
-	4-255	Reserved.

arg (16-bit unsigned char or unsigned short) indicates the service-specific operation to perform. The *arg* values for each service are defined in the SERVICE.H header file. The requester PAUSE and CONTINUE commands include the following options:

SYMBOLIC CONSTANT	VALUE	MEANING
SERVICE_CTRL_REDIR_DISK	1	Disk resource
SERVICE_CTRL_REDIR_PRINT	2	Print resource
SERVICE_CTRL_REDIR_COMM	4	Serial device

buf (unsigned char LSFAR *) points to the data structure described in [Service Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.

ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ServiceTableLocked	2180	The service does not respond to control actions.
NERR_ServiceNotInstalled	2184	The service has not been started.
NERR_ServiceCtlTimeout	2186	The service is not responding to the control function.
NERR_ServiceCtlBusy	2187	The service control is busy.
NERR_ServiceNotCtrl	2189	The service cannot be controlled in its present state.
NERR_ServiceKillProc	2190	The service was ended abnormally.
NERR_ServiceCtlNotValid	2191	The requested pause or stop is not valid for this service.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFlagProcess
- DosFsCtl
- DosGetInfoSeg
- DosGetShrSeg
- DosSemWait

Remarks

If the operation requested by the control *opcode* takes a long time to finish, the status and code values that the NetServiceControl API returns might be intermediate. Thus, for long-running operations, an application should issue successive calls to the NetServiceControl API to verify that the operation has been completed.

The NetServiceControl API acts only on services that are started. If a service is in the UNINSTALLED, UNINSTALL_PENDING, or INSTALL_PENDING state, the NetServiceControl API returns the NERR_ServiceCtlNotValid error code. There is one exception to this rule. An application can pass the *opcode* parameter with the value 0 (interrogation) to query the last known state of a stopped service. (If a service never has been started, the NetServiceControl API returns the NERR_ServiceNotInstalled error code.)

Services can be written to recognize a particular set of opcodes, as appropriate.

Related Information

For information about:

- Listing the services started on a server, see [Services - NetServiceEnum](#).
- Updating status and code information for a service, see [Services - NetServiceStatus](#).

NetServiceEnum or Net32ServiceEnum

NetServiceEnum or Net32ServiceEnum

The NetServiceEnum API retrieves information about all network services that are started.

Restrictions

This API can be called from DLS and OS/2 workstations. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <service.h>

NetServiceEnum(pszServername, sLevel, buf,
               usBuflen, pusEntriesReturned,
               pusEntriesAvail)           /*16 bit*/

Net32ServiceEnum(pszServername, ulLevel, buf,
                 ulBuflen, pulEntriesReturned,
                 pulEntriesAvail)        /*32 bit*/
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel/ or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, or 2, specifying which data structure to use, as described in [Service Information Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.

NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ServiceTableLocked	2180	The service does not respond to control actions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about updating the status and code information for a network service, see [Services - NetServiceStatus](#).

NetServiceGetInfo or Net32ServiceGetInfo

NetServiceGetInfo or Net32ServiceGetInfo

The NetServiceGetInfo API retrieves information about a particular network service that is started.

Restrictions

This API can be called from DLS and OS/2 workstations. This API does not have any access authority requirements.

Syntax

```
#include <netcons.h>
#include <service.h>

NetServiceGetInfo(pszServername, service,
                  sLevel, buf, usBuflen,
                  pusBytesAvail);          /* 16 bit*/

Net32ServiceGetInfo(pszServername, service,
                    ulLevel, buf, ulBuflen,
                    pulBytesAvail);       /* 32 bit*/
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

service (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the network service for which information is being requested.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, or 2, specifying which data structure to use, as described in [Service Information Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ServiceTableLocked	2180	The service does not respond to control actions.
NERR_ServiceNotInstalled	2184	The service has not been started.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

The NetServiceGetInfo API returns the NERR_Success code for services that are not started (SERVICE_UNINSTALLED). If a service is stopped, an application can examine the available data of a server using the NetServiceControl API.

The NetServiceGetInfo API is similar to the NetServiceControl API passed with the INTERROGATE opcode. The

NetServiceGetInfo API, however, does not interrogate the service; it only retrieves the status that the service last posted.

Related Information

For information about controlling the operations of a network service, see [Services - NetServiceControl](#).

NetServiceInstall or Net32ServiceInstall

NetServiceInstall or Net32ServiceInstall

The NetServiceInstall API starts a network service.

The name of the service is found in the IBMLAN.INI file. The executable file name of the service is matched to a corresponding entry in the Services section of the IBMLAN.INI file. Any relative file path name supplied for a service is assumed to be relative to the LAN Server root directory (IBMLAN).

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <service.h>

NetServiceInstall(pszServername, service, cmdargs,
                 buf, usBuflen);           /*16 bit*/

Net32ServiceInstall(pszServername, service, cmdargs,
                  buf, ulBuflen);         /*32 bit*/
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>service</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the network service to start.
<i>cmdargs</i>	<p>(const unsigned char LSFAR *) points to an ASCIIZ string containing the command parameters for <i>service</i>. The <i>cmdargs</i> parameter can be a null pointer or can point to a series of ASCIIZ string parameters ended by a NULL in the following example:</p> <pre>parm:value\0parm\0parm=value\0\0</pre> <p>The <i>cmdargs</i> parameters are merged with service component parameters from the IBMLAN.INI file and passed to the service program.</p>
<i>buf</i>	(unsigned char LSFAR *) points to the data structure described in Service Level 2 .

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_LanIniError	2131	An error occurred when opening or reading the IBMLAN.INI file.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_LineTooLong	2149	A line in the IBMLAN.INI file is too long.
NERR_ServiceTableLocked	2180	The service does not respond to control actions.
NERR_ServiceTableFull	2181	The service table is full.
NERR_ServiceInstalled	2182	The requested service has already been started.
NERR_ServiceEntryLocked	2183	The service does not respond to control actions.
NERR_BadServiceName	2185	The service name is not valid.
NERR_ServiceCtlTimeout	2186	The service is not responding to the control function.
NERR_ServiceCtlBusy	2187	The service control is busy.
NERR_BadServiceProgName	2188	The IBMLAN.INI file contains a service program name that is not valid.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosAllocShrSeg
- DosChgFilePtr
- DosExecPgm
- DosFsCtl

- DosGetShrSeg
- DosOpen
- DosRead
- DosSemWait
- DosStartSession

Related Information

For information about:

- Controlling network services, see [Services - NetServiceControl](#).
- Listing available servers, see [Server - NetServerEnum2](#).

NetServiceStatus or Net32ServiceStatus

NetServiceStatus or Net32ServiceStatus

The NetServiceStatus API sets status and code information for a network service.

Restrictions

A call to this API must be issued locally. This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <service.h>

NetServiceStatus(buf, ulBuflen);           /*32 bit*/
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

buf (unsigned char LSFAR *) points to the data structure to be passed with this API call, described in Service Status Data Structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_ServiceTableLocked	2180	The service does not respond to control actions.

NERR_ServiceNotInstalled 2184 The service has not been started.

Other codes could be returned from the following functions:

- DosGetInfoSeg
- DosGetShrSeg

Remarks

Service applications must call the NetServiceStatus API to update their status and code tables each time their status changes.

If a nonservice application (one not started by a call to the NetServiceInstall API) calls the NetServiceStatus API, the NetServiceStatus API returns the NERR_ServiceNotInstalled error code.

Related Information

For information about controlling the operation of a network service, see [Services - NetServiceControl](#).

Session Category

This category includes the following APIs:

[Session - NetSessionDel](#)
[Session - NetSessionEnum](#)
[Session - NetSessionGetInfo](#)

Sessions APIs control network sessions established between requesters and servers. They are used with the SHARES.H and NETCONS.H header files.

A *session* is a path between a requester and a server. A requester begins a session with a server the first time the requester tries to connect to a shared resource on the server. Any further connections from the requester to the other shared resources on the same server do not create another session; multiple connections can be serviced on one session.

To end a session, an application calls the NetSessionDel function. This action deletes all current connections between the requester and the server.

The NetSessionEnum function returns information about all sessions established with a server.

To obtain information about a particular session, an application calls the NetSessionGetInfo function.

Session Data Structures

The *sLevel* parameter controls the level of information that the NetSessionEnum and NetSessionGetInfo APIs return.

Session Level 0

```
struct session_info_0 {  
    unsigned char LSFAR * LSPTR    sesi0_cname;  
};
```

where *sesi0_cname* points to an ASCIIZ string containing the computer name of the requester that established the session.

Session Level 1

```
struct session_info_1 {
    unsigned char LSFAR * LSPTR    sesi1_cname;
    unsigned char LSFAR * LSPTR    sesi1_username;
    unsigned short                sesi1_num_conns;
    unsigned short                sesi1_num_opens;
    unsigned short                sesi1_num_users;
    unsigned long                 sesi1_sess_time;
    unsigned long                 sesi1_idle_time;
    unsigned long                 sesi1_user_flags;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *sesi1_username* points to an ASCIIZ string containing the name of the user who established the session.
- *sesi1_num_conns* indicates the number of connections that have been made during the session.
- *sesi1_num_opens* indicates the number of files, devices, and pipes that have been opened during the session.
- *sesi1_num_users* specifies the number of sessions that are established between the server and the requester. This value is always 0 or 1.
- *sesi1_sess_time* indicates the number of seconds a session has been active.
- *sesi1_idle_time* indicates the number of seconds a session has been idle.
- *sesi1_user_flags* indicates the manner in which the user established the session. The bit mask for *sesi1_user_flags* is defined as follows in SHARES.H:

SYMBOLIC CONSTANT	VALUE	MEANING
SESS_GUEST	1	sesi1_username established the session using a GUEST account.
SESS_NOENCRYPTION	2	sesi1_username established the session without using password encryption.

Session Level 2

```
struct session_info_2 {
    unsigned char LSFAR * LSPTR    sesi2_cname;
    unsigned char LSFAR * LSPTR    sesi2_username;
    unsigned short                sesi2_num_conns;
    unsigned short                sesi2_num_opens;
    unsigned short                sesi2_num_users;
};
```

```

unsigned long          sesi2_sess_time;
unsigned long          sesi2_idle_time;
unsigned long          sesi2_user_flags;
unsigned char LSFAR * LSPTR sesi2_cltype_name;
};

```

where:

- The first eight fields in this data structure are identical to those in the previous level.
- *sesi2_cltype_name* specifies the type of client that established the session. The following types are defined in SHARES.H:

TYPE	MEANING
Down Level	Old clients; for example, LAN Server 1.0 clients
DLR 2.0	DLR 2.0 clients
OS/2 LS 1.3	OS/2 LAN Server 1.3 clients
OS/2 LS 3.0	OS/2 LAN Server 3.0 clients
OS/2 LS 4.0	OS/2 LAN Server 4.0 clients

Session Level 10

```

struct session_info_10 {
    unsigned char LSFAR * LSPTR sesi10_cname;
    unsigned char LSFAR * LSPTR sesi10_username;
    unsigned long sesi10_sess_time;
    unsigned long sesi10_idle_time;
};

```

where All the fields in this data structure are identical to those in the previous level.

NetSessionDel or Net32SessionDel

NetSessionDel or Net32SessionDel

The NetSessionDel API ends a session between a requester and a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```

#include <netcons.h>
#include <shares.h>

```

```
NetSessionDel(pszServername, pszWorkstation,
              reserved);          /* 16 bit */

Net32SessionDel(pszServername, pszWorkstation,
                reserved);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszWorkstation (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the requester that established the session that is being discontinued.

reserved (16-bit short or 32-bit unsigned long) must be 0.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ClientNameNotFound	2312	A session does not exist with that computer name.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_NoSuchServer	2460	The server ID is not valid.
NERR_NoSuchSession	2461	The session ID is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetProcAddress
- DosGetShrSeg
- DosLoadModule

Remarks

All connections established during the session are disconnected when the session is deleted, and any files that were opened during the session are closed. Data may be lost if any process on the requester is communicating with the server when NetSessionDel is called.

Using the NetSessionDel command for the session that is running on the requester from which the command is issued does not result in the NERR_Success message, even if the operation is successful. DOS users receive a return code of 59, ERROR_UNEXP_NET_ERR. OS/2 users receive a return code of 240, ERROR_VC_DISCONNECTED. Issuing the NetSessionDel command for a session that is running on another requester returns the NERR_Success error code if the command is run successfully.

Related Information

For information about retrieving the status of the session of a server, see [Session - NetSessionGetInfo](#).

NetSessionEnum or Net32SessionEnum

NetSessionEnum or Net32SessionEnum

The NetSessionEnum API provides information about all current sessions to a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. Users with Server operator privileges can receive session information at level 0 or level 10. Administrator authority is required for full access.

Syntax

```
#include <netcons.h>
#include <shares.h>

NetSessionEnum(pszServername, sLevel,
               buf, usBuflen, pusEntriesReturned,
               pusEntriesAvail);          /* 16 bit */

Net32SessionEnum(pszServername, ulLevel,
                 buf, ulBuflen, pulEntriesReturned,
                 pulEntriesAvail);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) specifies the level of detail (0, 1, 2, or 10) requested for the returned *session_info* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of

return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsRamSemClear
- DosFreeSeg
- DosFsCtl
- DosGetShrSeg
- DosSemClear

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Deleting a session, see [Session - NetSessionDel](#).
- Listing all sessions redirected to a resource, see [Connection - NetConnectionEnum](#).

NetSessionGetInfo or Net32SessionGetInfo

NetSessionGetInfo or Net32SessionGetInfo

The NetSessionGetInfo API retrieves information about a session established between a particular requester and server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. Users with Server operator privilege can receive session information at level 0 or level 10. Administrator authority is required for full access.

Syntax

```
#include <netcons.h>
#include <shares.h>

NetSessionGetInfo(pszServername, pszWorkstation,
                  sLevel, buf, usBuflen,
                  pusBytesAvail);          /* 16 bit */

Net32SessionGetInfo(pszServername, pszWorkstation,
                    ulLevel, buf, ulBuflen,
                    pulBytesAvail);       /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- pszWorkstation* (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the requester whose session is to be monitored.
- sLevel* or *ulLevel* (16-bit short or 32-bit unsigned long) specifies the level of detail (0, 1, 2, or 10) requested for the returned *session_info* data structure.
- buf* (unsigned char LSFAR *) points to the *session_info* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.

NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_ClientNameNotFound	2312	A session does not exist with that computer name.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about listing all sessions redirected to a resource, see [Connection - NetConnectionEnum](#).

Share Category

This category includes the following APIs:

[Share - NetShareAdd](#)
[Share - NetShareCheck](#)
[Share - NetShareDel](#)
[Share - NetShareEnum](#)
[Share - NetShareGetInfo](#)
[Share - NetShareSetInfo](#)

Share APIs control shared resources. They are used with the SHARES.H, ACCESS.H, and NETCONS.H header files.

Share is the term applied to the local device of a server (such as a disk drive, print device, or named pipe) that other applications on the network can access. A unique *netname* is assigned to each shared resource to enable remote users and applications to refer to the share, rather than the local device name of the share.

The first step for allowing remote users and applications to access a server resource is to share the resource, giving it a netname. This is done with the NetShareAdd function, which adds a share to a server. The function requires information about the resource type.

On a server, NetShareAdd requires only a netname and a local device name to share a resource. A user or an application must have an account on the server to access the resource.

DOS Considerations

Under DOS, the NetShareAdd, NetShareDel, and NetShareEnum APIs can be run on or called from a DLS workstation if DLS Peer service is installed. The user level security function for DOS LAN Services allows a DLS peer workstation to run similar to an OS/2 server or an OS/2 peer workstation. In this mode, resources are shared by the DLS peer workstation based on access rights an administrator assigns to an individual user or a group of users.

The rest of the APIs in this category can be run only on a remote server and, if called locally on a DLS requester, the NERR_RemoteOnly error code is returned.

Share Data Structures

Share information can be returned by the NetShareEnum and NetShareGetInfo functions at one of three levels of detail specified by the *sLevel* parameter (values 0, 1, or 2). NetShareAdd requires level 2 of detail. NetShareSetInfo can be called with either level 1 or level 2. NetShareCheck and NetShareDel do not use or return data structures.

The following data structures are associated with *sLevel* values 0, 1, and 2.

Share Level 0

```
struct share_info_0 {
    unsigned char shi0_netname[NNLEN+1];
};
```

where *shi0_netname* is an ASCIIZ string containing the netname of a resource.

Share Level 1

```
struct share_info_1 {
    unsigned char          shi1_netname[NNLEN+1];
    unsigned char          shi1_pad1;
    unsigned short         shi1_type;
    unsigned char LSFAR * LSPTR shi1_remark;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *shi1_pad1* word-aligns the data structure components.
- *shi1_type* is one of the following four values, defined in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	SHARE TYPE
STYPE_DISKTREE	0	Disk drive
STYPE_PRINTQ	1	Spooler queue

STYPE_DEVICE	2	Serial device
STYPE_IPC	3	Interprocess communication (IPC)

- *shi1_remark* points to an ASCIIZ string containing an optional comment about the shared resource.

Share Level 2

```
struct share_info_2 {
    unsigned char          shi2_netname[NNLEN+1];
    unsigned char          shi2_pad1;
    unsigned short         shi2_type;
    unsigned char LSFAR * LSPTR shi2_remark;
    unsigned short         shi2_permissions;
    unsigned short         shi2_max_uses;
    unsigned short         shi2_current_uses;
    unsigned char LSFAR * LSPTR shi2_path;
    unsigned char          shi2_passwd[SHPWLEN+1];
    unsigned char          shi2_pad2;
};
```

where:

- The first two fields in this data structure are identical to those in the previous level.
- *shi2_type* is one of four values indicating the type of share, as defined in the SHARES.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
STYPE_DISKTREE	0	Disk drive
STYPE_PRINTQ	1	Spooler queue
STYPE_DEVICE	2	Serial device
STYPE_IPC	3	Interprocess communication (IPC)

Note: The *shi2_type* value affects the requirements for certain other *share_info_2* components when the NetShareAdd function is called. See "Remarks" in topic [Share - NetShareAdd](#) for details.

- The next field is identical to the same field in the previous level.
- *shi2_permissions* is used only for DLS applications and ignored for OS/2 applications. The meaning of this field depends on which type of resource is in query (specified by the *shi1_type* field).
 - If *shi1_type* is STYPE_DISKTREE, this field can be any combination of the following permissions, except for ACCESS_READ + ACCESS_CREATE. These are defined for use with DLS applications in the ACCESS.H header file as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
ACCESS_READ	0x01	Permission to read data from a resource.
ACCESS_WRITE	0x02	Permission to write data to a resource.
ACCESS_CREATE	0x04	Permission to create an instance of the resour

(such as a file). Data can be written to the resource as the resource is created.

- If *shi1_type* is STYPE_PRINTQ, the *shi2_permissions* field specifies the form control at the end of a print job. These are defined for use with DLS applications in the DOSPRINT.H header file as follows:

SYMBOLIC CONSTANT	BIT VALUE	MEANING
FORMFEED_NO	1	The printer will not be sent a command at the end of any print job to position to the top of the page.
FORMFEED_AUTO	2	An attempt is made to determine whether a form feed is needed for a print job. If the last 6 bytes of the file contain no form feed command or follow a form feed command with printable characters, a form feed command is issued to printer.
FORMFEED_YES	5	A form feed command is issued to the printer at the end of each print job to position to the top of the next form (default).

Note: DLS does not support sharing of serial devices (STYPE_DEVICE). In some circumstances, however, COM devices, such as COM1, can be shared as print devices (STYPE_PRINTQ); for example, if operating in synchronous communication mode, such as with a serial printer.

- *shi2_max_uses* gives the maximum number of concurrent connections the shared resource can accommodate. This number is unlimited if the *shi2_max_uses* value is -1. (This field is ignored for DLS.)
- *shi2_current_uses* indicates the number of connections currently made to the resource. (This field is ignored for DLS.)
- *shi2_path* points to an ASCIIZ string containing the local path name of the shared resource. For disks, *shi2_path* is the path being shared. For spooler queues, *shi2_path* is the name of the spooler queue being shared. For serial device queues, *shi2_path* is a string of one or more communication device names separated by spaces; for example, COM1 COM2 COM6. The maximum length of the string is 254 bytes.
- *shi2_passwd* is a reserved field and must be NULL.
- *shi2_pad2* word-aligns the data structure component.

Related Information

For information about:

- Setting up user accounts, see [User Category](#).
- Assigning access permissions, see [Access Permission Category](#).

NetShareAdd or Net32ShareAdd

NetShareAdd or Net32ShareAdd

The NetShareAdd API shares the resource of a server.

Restrictions

This API can be called from DLS and OS/2 workstations. DLS workstations can issue this call locally only if the Peer service is started. A call to this API must be issued to a server or to a workstation with the Peer service installed. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <shares.h>
#include <access.h>

NetShareAdd(pszServername, sLevel,
            buf, usBuflen);    /* 16 bit */

Net32ShareAdd(pszServername, ulLevel,
              buf, ulBuflen); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 2, which specifies the data structure described in [Share Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_FILENAME_EXCED_RANGE	206.	The file name is longer than 8 characters or the extension is longer than characters.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_UnknownServer	2103	The server cannot be located.
NERR_ServerNotStarted	2114	The Server service has not been

		started.
NERR_UnknownDevDir	2116	The device or directory does not exist.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_DuplicateShare	2118	The name has already been shared.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_QNotFound	2150	The printer queue does not exist.
NERR_DeviceShareConflict	2318	This device cannot be shared as both a spooled and a non-spooled device.
NERR_BadDevString	2340	This list of devices is not valid.
NERR_BadDev	2341	The requested device is not valid.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CantType	2357	The type of input cannot be determined.

Other codes could be returned from the following functions:

- DosDevIOCtl
- DosFsRamSemRequest
- DosFsCtl
- DosGetProcAddr
- DosGetShrSeg
- DosLoadModule
- DosPrintDestGetInfo
- DosSemRequest

Remarks

Depending on the type of share specified by the value in the *shi2_type* component of the *share_info_2* data structure, other components in the data structure must be specified as follows:

SYMBOLIC CONSTANT	VALUE	COMPONENT REQUIREMENTS
STYPE_DISKTREE	0	<i>shi2_path</i> must specify a file-system path name
STYPE_PRINTQ	1	<i>shi2_path</i> must specify the name of an existing spooler queue. It is recommended that the netname and the path be the same; otherwise, the results are unpredictable.
STYPE_DEVICE	2	<i>shi2_path</i> must either be passed as a null

STYPE_IPC	3	shi2_netname must specify a shared interprocess communication resource, and shi2_path must point to a null string.
-----------	---	--

Related Information

- Removing a list of shareable resources, see [Share - NetShareDel](#).
- Listing the shareable resources of a server, see [Share - NetShareEnum](#).

NetShareCheck or Net32ShareCheck

Restrictions

Syntax

Parameters

<i>devname</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the device to be checked.
<i>type</i>	(16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) indicates the type of shared device, as defined in the SHARES.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
STYPE_DISKTREE	0	Disk drive
STYPE_PRINTQ	1	Spooler queue

STYPE_DEVICE	2	Serial device
STYPE_IPC	3	Interprocess communication device

The returned *type* value is valid only if NetShareCheck is successful.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_DeviceNotShared	2311	This device is not shared.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_SourceIsDir	2380	The source path cannot be a directory.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about:

- Reconfiguring the shareable resource of a server, see [Share - NetShareSetInfo](#).
- Retrieving the status of a shared resource, see [Share - NetShareGetInfo](#).

NetShareDel or Net32ShareDel

The NetShareDel API deletes a netname from the list of shared resources of a server.

When this function deletes a netname, all connections to the shared resource are disconnected.

Restrictions

This API can be called from DLS and OS/2 workstations. DLS workstations can issue this call locally only if the Peer service is started. A call to this API must be issued to a server or to a workstation with the Peer service installed. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <shares.h>
#include <access.h>

NetShareDel(pszServername, netname,
            reserved);    /* 16 bit */

Net32ShareDel(pszServername, netname,
              reserved);  /* 32bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

netname (const unsigned char LSFAR *) points to an ASCIIZ string specifying the netname to be deleted.

reserved (16-bit unsigned short or 32-bit unsigned long) must be 0.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.

NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetProcAddr
- DosGetShrSeg
- DosLoadModule

Related Information

For information about:

- Adding a share on a server, see [Share - NetShareAdd](#).
- Listing all connections to a shared resource, see [Connection - NetConnectionEnum](#).
- Listing the shareable resources of a server, see [Share - NetShareEnum](#).

NetShareEnum or Net32ShareEnum

NetShareEnum or Net32ShareEnum

The NetShareEnum API retrieves share information about each shared resource on a server.

Restrictions

This API can be called from DLS and OS/2 workstations. DLS workstations can issue this call locally only if the Peer service is started. A call to this API must be issued to a server or to a workstation with the Peer service installed. User authority is allowed limited access to this API. Administrator authority is required for full access.

If the *sLevel* parameter of NetShareEnum is passed with the value 2, the calling application must have administrative privileges on the server.

Syntax

```
#include <netcons.h>
#include <shares.h>
#include <access.h>

NetShareEnum(pszServername, sLevel, buf, usBuflen,
             pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32ShareEnum(pszServername, ulLevel, buf, ulBuflen,
               pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, or 2, specifying which data structure to use as described in [Share Level 0](#) through [Share Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about retrieving information about a particular shared resource, see [Share - NetShareGetInfo](#).

NetShareGetInfo or Net32ShareGetInfo

NetShareGetInfo or Net32ShareGetInfo

The NetShareGetInfo API retrieves information about a particular shared resource on a server.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. A call to this API can be issued only to an OS/2 LAN Server workstation. User authority is allowed limited access to this API. Administrator authority is required for full access.

If the *sLevel* parameter of NetShareGetInfo is passed with the value 2, the calling application must have administrative privileges on the server.

Syntax

```
#include <netcons.h>
#include <shares.h>
#include <access.h>

NetShareGetInfo(pszServername, netname, sLevel, buf,
               usBuflen, pusBytesAvail);    /* 16 bit */

Net32ShareGetInfo(pszServername, netname, ulLevel, buf,
                 ulBuflen, pulBytesAvail);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- netname* (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the share of interest.
- sLevel* or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, or 2, specifying which data structure to use, as described in [Share Level 0](#) through [Share Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-

length data.

NERR_RemoteErr	2127	A remote API error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about reconfiguring a shareable server resource, see [Share - NetShareSetInfo](#).

NetShareSetInfo or Net32ShareSetInfo

NetShareSetInfo or Net32ShareSetInfo

The NetShareSetInfo API sets the parameters of a shared resource.

In addition to using levels 1 and 2 of the *share_info* data structure, a call to this API can specify particular fields within those data structures. The advantage to SetInfo APIs is that they often require less program code and use less local workstation memory. (The amount of memory space, specified by the *buf* and *usBuflen* parameters, only has to be large enough to contain the single field from the data structure rather than the entire data structure.)

The *parmnum* parameter specifies whether to pass the entire data structure, or which field from the data structure is to be passed.

Restrictions

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. A call to this API can be issued only to an OS/2 LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <shares.h>
#include <access.h>

NetShareSetInfo(pszServername, netname, sLevel,
               buf, usBuflen, parmnum);    /* 16 bit */
```

```
Net32ShareSetInfo(pszServername, netname, ulLevel,
                  buf, ulBuflen, parmnum); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>netname</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the netname of the resource to be set.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 1 or 2, specifying which data structure to use, as described in Share Level 1 and Share Level 2 .
<i>parmnum</i>	(16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a specific field in the data structure is to be passed. If the value is 0, the entire data structure is sent. Otherwise, <i>parmnum</i> specifies which field in the data structure is to be sent. (Specify the field by ordinal position, where 1 passes the first field, and so on.) For this API, <i>parmnum</i> can be any one of the following:

SYMBOLIC CONSTANT	VALUE	DATA STRUCTURE FIELD
SHI_REMARK_PARMNUM	4	shi1_remark or shi2_remark
SHI_MAX_USES_PARMNUM	6	shi2_max_uses

NOTE: If you specify parmnum to be 0, you must allocate enough (specified by buf and ulBuflen) to contain the entire data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.

NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg

Related Information

For information about retrieving the status of a shareable server resource, see [Share - NetShareGetInfo](#).

Spooler Category

Spooler APIs give an application access to spooler queue manager operations. These functions are provided by the base operating system and supported by LAN Server across the network.

The following table summarizes the functions used to print jobs, manage jobs, and manipulate the spooler function. They are defined in the base operating system header file, PMSPL.H. For more information, see the *OS/2 Technical Library, Programming Guide Volume 3*.

FUNCTION NAME	DESCRIPTION
DevCloseDC	Closes a device context
DevEscape	Controls print job start, new page, and end
DevOpenDC	Opens a device context
DevPostDeviceModes	Displays the job properties dialog or retrieves device job property defaults
DevQueryCaps	Retrieves information about the device's capabilities
DevQueryHardcopyCaps	Retrieves information about forms

SplControlDevice	Cancels, holds, continues, or restarts a printer device
SplCopyJob	Copies a print job (within the same printer queue)
SplCreateDevice	Creates a printer device
SplCreateQueue	Creates a printer queue
SplDeleteDevice	Deletes a printer device
SplDeleteJob	Deletes a print job
SplDeleteQueue	Deletes a queue
SplEnumDevice	Lists printer devices, optionally, with status information
SplEnumDriver	Lists printer drivers
SplEnumJob	Lists print jobs
SplEnumPort	Lists physical ports
SplEnumPrinter	Lists printer queues and print devices, optionally, for the whole network
SplEnumQueue	Lists printer queues
SplEnumQueueProcessor	Lists queue processors (queue drivers)
SplHoldJob	Holds a job in a printer queue
SplHoldQueue	Holds a printer queue
SplQmAbort	Stops a print job and closes a spooler session
SplQmAbortDoc	Stops a print job
SplQmClose	Closes a spooler session
SplQmEndDoc	Ends a print job
SplQmOpen	Opens a spooler session
SplQmStartDoc	Starts a print job
SplQmWrite	Writes data into a print job
SplQueryDevice	Retrieves information about a printer device
SplQueryJob	Retrieves information about a print job
SplQueryQueue	Retrieves information about a printer queue
SplReleaseJob	Releases a print job in a printer queue
SplSetDevice	Sets information about a printer device
SplSetJob	Sets information about a print job
SplSetQueue	Sets information about a printer queue

For information about print, spool, and other base operating system functions, see the online information included with your *IBM OS/2 Programming Toolkit*.

If you are programming DOS or 16-bit OS/2 applications and already have any of the following, you might find them to be helpful references. (LAN Server Version 4.0 offers limited support to the APIs described in these products.)

- *IBM OS/2 Version 1.3 Programming Guide Technical Update*
- *IBM Presentation Manager Programming Reference Version 1.3 Technical Upgrade Volume 1*
- *IBM Presentation Manager Programming Reference Version 1.3 Technical Upgrade Volume 2*
- Online programming information included with the *IBM OS/2 Programming Toolkit Version 1.3*

Spooler API Equivalents to DOS Print APIs

For OS/2, the header information for the spooler APIs is defined in the PMSPL.H header file, in the OS/2 Toolkit.

Depending on the files that are included and the definitions that are used, there can be either 16-bit or 32-bit support for print applications. If you want to take a program written for a 16-bit OS/2 application and recompile it for 32-bit OS/2, you must use PMSPL.H from the Toolkit.

The following are spooler API equivalents to the DOS print APIs:

FOR 16-BIT SUPPORT	FOR 32-BIT SUPPORT
DosPrintDestControl	SplControlDevice
DosPrintDestGetInfo	SplQueryDevice
DosPrintDestEnum	SplEnumDevice
DosPrintJobContinue	SplReleaseJob
DosPrintJobPause	SplHoldJob
DosPrintJobDel	SplDeleteJob
DosPrintJobGetInfo	SplQueryJob
DosPrintJobSetInfo	SplSetJob
DosPrintJobEnum	SplEnumJob
DosPrintQAdd	SplCreateQueue
DosPrintQPause	SplHoldQueue
DosPrintQContinue	SplReleaseQueue
DosPrintQDel	SplDeleteQueue
DosPrintQGetInfo	SplQueryQueue
DosPrintQSetInfo	SplSetQueue
DosPrintQEnum	SplEnumQueue
DosPrintDestAdd	SplCreateDevice
DosPrintDestSetInfo	SplSetDevice

DosPrintDestDel	SplDeleteDevice
DosPrintQPurge	SplPurgeQueue
DosPrintDriverEnum	SplEnumDriver
DosPrintQProcessorEnum	SplEnumQueueProcessor
DosPrintPortEnum	SplEnumPort

Statistics Category

This category includes the following API:

[Statistics - NetStatisticsGet2](#)

Statistics APIs retrieve and clear the operating statistics for requesters and servers. These APIs are used with the NETSTATS.H and NETCONS.H header files.

LAN Server accumulates a set of operating statistics for requesters and servers from the time a requester or server service is started. To retrieve the statistics, call the NetStatisticsGet2 function.

The NetStatisticsGet2 API returns the statistics in the following data structures.

Requester Statistics Data Structure

```
struct stat_workstation_0 {
    unsigned long    stw0_start;
    unsigned long    stw0_numNCB_r;
    unsigned long    stw0_numNCB_s;
    unsigned long    stw0_numNCB_a;
    unsigned long    stw0_fiNCB_r;
    unsigned long    stw0_fiNCB_s;
    unsigned long    stw0_fiNCB_a;
    unsigned long    stw0_fcNCB_r;
    unsigned long    stw0_fcNCB_s;
    unsigned long    stw0_fcNCB_a;
    unsigned long    stw0_sesstart;
    unsigned long    stw0_sessfailcon;
    unsigned long    stw0_sessbroke;
    unsigned long    stw0_uses;
    unsigned long    stw0_usefail;
    unsigned long    stw0_autorec;
    unsigned long    stw0_bytessent_r_hi;
    unsigned long    stw0_bytessent_r_lo;
    unsigned long    stw0_bytesrcvd_r_hi;
    unsigned long    stw0_bytesrcvd_r_lo;
    unsigned long    stw0_bytessent_s_hi;
    unsigned long    stw0_bytessent_s_lo;
    unsigned long    stw0_bytesrcvd_s_hi;
    unsigned long    stw0_bytesrcvd_s_lo;
    unsigned long    stw0_bytessent_a_hi;
    unsigned long    stw0_bytessent_a_lo;
    unsigned long    stw0_bytesrcvd_a_hi;
    unsigned long    stw0_bytesrcvd_a_lo;
    unsigned long    stw0_reqbufneed;
    unsigned long    stw0_bigbufneed;
};
```

where:

- *start* is the time statistics collection started. This field indicates the date or time that the statistics were last cleared; that is, it indicates the time period over which the returned statistics were collected.
- *numNCB* fields indicate the number of network control blocks (NCBs) issued from each source and include failed NCBs. To get the total number of successful NCBs issued, subtract the number of failed NCBs. These numbers are held as follows in the *fcNCB* fields:
 - *numNCB_r* is the number of NCBs issued (redirector).
 - *numNCB_s* is the number of NCBs issued (server).
 - *numNCB_a* is the number of NCBs issued (application).
- *fiNCB* fields indicate the number of NCBs that failed at the time they were issued, for whatever reason. These counts are included in the total number of NCBs issued, as follows:
 - *fiNCB_r* is the number of NCBs that failed issue (redirector).
 - *fiNCB_s* is the number of NCBs that failed issue (server).
 - *fiNCB_a* is the number of NCBs that failed issue (application).
- *fcNCB* fields indicate the number of NCBs that failed after issue, whether at or before completion. These counts are included in the total number of NCBs issued, as follows:
 - *fcNCB_r* is the number of NCBs that failed completion (redirector).
 - *fcNCB_s* is the number of NCBs that failed completion (server).
 - *fcNCB_a* is the number of NCBs that failed completion (application).
- *sesstart* is the number of requester sessions started.
- *sessfailcon* is the number of requester session failures to connect, except those that failed because of name not found.
- *sessbroke* is the number of failures of requester sessions, after the sessions were established.
- *uses* is the number of requester uses.
- *usefail* is the number of requester use failures. This is a count of failed tree-connects, when a server is found but the resources were not found.
- *autorec* is the number of requester autoconnects.
- The following 12 fields form six quadwords, which contain very large counters. The high doubleword of each is the value divided by 2³², while the low doubleword is the value modulo 2³². A quadword is a data area that is twice as large as a doubleword.

These fields count total bytes in all NCBs sent and received for all three categories: issued, failed issue, and failed completion. Server information is included to provide an accurate total.

Note: For all the NCB-related and bytes-count counters:

Those with the suffix *_r* are from the redirector. These NCBs are issued by the redirector as part of the normal process of maintaining remote network connections.

Those with the suffix *_s* are server-related. These NCBs are sent by the redirector on behalf of the server to maintain shared resource connections.

Those with the suffix *_a* are application-generated. These NCBs can be caused by applications calling NetBiosSubmit, the use of second-class mailslots, server announcements (sending and receiving), and so on.

- *bytessent_r_hi* is the number of requester bytes sent to the network (high doubleword).
- *bytessent_r_lo* is the number of requester bytes sent to the network (low doubleword).
- *bytesrcvd_r_hi* is the number of requester bytes received from the network (high doubleword).
- *bytesrcvd_r_lo* is the number of requester bytes received from the network (low doubleword).
- *bytessent_s_hi* is the number of server bytes sent to the network (high doubleword).
- *bytessent_s_lo* is the number of server bytes sent to the network (low doubleword).
- *bytesrcvd_s_hi* is the number of server bytes received from the network (high doubleword).
- *bytesrcvd_s_lo* is the number of server bytes received from the network (low doubleword).

- *bytessent_a_hi* is the number of application bytes sent to the network (high doubleword).
- *bytessent_a_lo* is the number of application bytes sent to the network (low doubleword).
- *bytesrcvd_a_hi* is the number of application bytes received from the network (high doubleword).
- *bytesrcvd_a_lo* is the number of application bytes received from the network (low doubleword).
- *reqbufneed* is the number of times the requester required a request buffer but failed to allocate one. This indicates that the parameters of the requester might need adjustment.
- *bigbufneed* is the number of times the requester required a big buffer but failed to allocate one. This indicates that the parameters of the requester might need adjustment.

Note: A value of -1 or 0xFFFFFFFF for any field means that information is not available. A value of -2 or 0xFFFFFFFFE means that the field has overflowed.

Server Statistics Data Structure

```
struct stat_server_0 {
    unsigned long    sts0_start;
    unsigned long    sts0_fopens;
    unsigned long    sts0_devopens;
    unsigned long    sts0_jobsqueued;
    unsigned long    sts0_sopens;
    unsigned long    sts0_stimedout;
    unsigned long    sts0_serrorout;
    unsigned long    sts0_pwerrors;
    unsigned long    sts0_permerrors;
    unsigned long    sts0_syserrors;
    unsigned long    sts0_bytessent_high;
    unsigned long    sts0_bytessent_low;
    unsigned long    sts0_bytesrcvd_high;
    unsigned long    sts0_bytesrcvd_low;
    unsigned long    stw0_avresponse;
    unsigned long    stw0_reqbufneed;
    unsigned long    stw0_bigbufneed;
};
```

where:

- *start* is the time statistics collection started. This field indicates the date and time that the statistics were last cleared; that is, it indicates the time period over which the returned statistics were collected.
- *fopens* is the number of server file opens. This includes opens of named pipes.
- *devopens* is the number of server device opens.
- *jobsqueued* is the number of server print jobs spooled.
- *sopens* is the number of server session starts.
- *stimedout* is the number of server session autotimeouts.
- *serrorout* is the number of server sessions errored out.
- *pwerrors* is the number of server password violations.
- *permerrors* is the number of server access permission errors.
- *syserrors* is the number of server system errors.
- The following 4 fields form two quadwords that contain very large counters. The high doubleword of each is the value divided by

2[32], while the low doubleword is the value modulo 2[32]:

- *bytessent_high* is the number of server bytes sent to the network (high doubleword).
 - *bytessent_low* is the number of server bytes sent to the network (low doubleword).
 - *bytesrcvd_high* is the number of server bytes received from the network (high doubleword).
 - *bytesrcvd_low* is the number of server bytes received from the network (low doubleword).
- *avresponse* is the average server response time in milliseconds.
 - *reqbufneed* is the number of times the server required a request buffer but failed to allocate one. This indicates that the parameters of the server might need adjustment.
 - *bigbufneed* is the number of times the server required a big buffer but failed to allocate one. This indicates that the parameters of the server might need adjustment.

Note: A value of -1 or 0xFFFFFFFF for any field means that information is not available. A value of -2 or 0xFFFFFFFFE means that the field has overflowed.

NetStatisticsGet2 or Net32StatisticsGet2

NetStatisticsGet2 or Net32StatisticsGet2

The NetStatisticsGet2 API retrieves and optionally clears operating statistics for a service.

Note: The NetStatisticsGet2 API replaces both the NetStatisticsGet API and the NetStatisticsClear API, which are now obsolete.

Restrictions

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API.

Syntax

```
#include <netcons.h>
#include <netstats.h>

NetStatisticsGet2(pszServername, servicename, reserved,
                 sLevel, options, buf,
                 usBuflen, pusBytesAvail);    /* 16 bit */

Net32StatisticsGet2(pszServername, servicename, reserved,
                   ulLevel, options, buf,
                   ulBuflen, pulBytesAvail);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>servicename</i>	(const unsigned char LSFAR *) is the ASCIIZ service name for which to get the statistics. Only SERVER and REQUESTER are allowed for <i>servicename</i> . Other names produce the ERROR_NOT_SUPPORTED error code. If the server statistics are requested and the server is not running, the error code returned is NERR_ServiceNotInstalled.
<i>reserved</i>	(unsigned long) must be 0.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) specifies the level of detail returned by either the *stat_workstation_0* or the *stat_server_0* data structure.

The valid values for *sLevel* depend on the value of *servicename*. The current valid value for REQUESTER or SERVER is 0.

options (unsigned long) are the options flags.

The *options* parameter is bitmapped as follows:

BIT	MASK	SYMBOL	MEANING
0	0x1	STATSOPT_CLR	Clear statistics after retrieval.
1-31			Must be 0.

The option to clear the statistics allows automatic Get and Clear operations, which allows an application that is compiling cumulative numbers to make sure no data slips by in the time between the Get operation and the Clear operation.

buf (unsigned char LSFAR *) points to either the returned *stat_workstation_0* or the *stat_server_0* data structure.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ServiceNotInstalled	2184	The service has not been started.
NERR_BadServiceName	2185	The service name is not valid.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosGetShrSeg
- DosSemClear

Remarks

This API returns a full data structure. As with other GetInfo calls, the error NERR_BuffTooSmall is returned if the supplied buffer is too small for the required data.

If the service name specified is REQUESTER, then *stat_workstation_0* is returned to *but*. If the service name specified is SERVER, then *stat_server* is returned to *but*.

Use Category

This category includes the following APIs:

[Use - NetUseAdd](#)
[Use - NetUseDel](#)
[Use - NetUseEnum](#)
[Use - NetUseGetInfo](#)

Use APIs to examine or control connections (uses) between requesters and servers. Administrative privilege is required to call them remotely. They are used with the USE.H and NETCONS.H header files.

The NetUseAdd function establishes a connection between a local computer and a resource shared on a server by redirecting a null or local device name to a shared resource on that remote server. The following types of connections can be made:

- Device name connections, which only can be explicit
- Universal naming convention (UNC) connections, which can be either explicit or implicit

To establish an explicit device name connection, NetUseAdd redirects a local device name to the netname of a remote server resource (*\\servername\netname*). Once a device name connection is made, users or applications can use the remote resource by specifying the local device name.

UNC connections can be explicit, created by the NetUseAdd function, or implicit, made by way of an OS/2 function (responsible for the connection).

To establish an explicit UNC connection, NetUseAdd redirects a null device name to the netname of a remote server resource.

To establish an implicit UNC connection, an application passes the netname of the resource to any one of the OS/2 functions that accept netnames (such as the DosOpen function). The UNC name is understood by the OS/2 function and a connection is made to the specified netname. All further requests on this connection require the full netname.

Note: Connections are to be distinguished from *sessions*. A session is a path between a requester and a server, established the first time a requester makes a connection with one of the shared resources of the server. All further connections between the requester and the

server are part of this same session until the session is ended by calling the NetSessionDel function.

The following examples show how the three types of connections work:

- Explicit device name connection. Assume an application redirected the local device name *d:* to the netname \\DEVELOP\\SRCDRV by calling the NetUseAdd function as shown:

```
strncpy (buf.uil_local, "d:", 3);
retcode = NetUseAdd(NULL, 1, BUF, BUFLLEN);
```

To access files on this resource, an application need only specify the redirected device name and the name of the file, as shown:

```
retcode = system("type d:\\read.me");
```

- Explicit UNC connection. Assume an application redirected a null device name to the remote resource \\DEVELOP\\SRCDRV by calling the NetUseAdd function, as shown:

```
strncpy (buf.uil_local, "", 3);
retcode = NetUseAdd(NULL, 1, BUF, BUFLLEN);
```

To display the contents of the READ.ME file on the resource \\DEVELOP\\SRCDRV, an application also can specify only the name of the resource with the following command:

```
retcode=system("type \\\\develo\\srcdrv\\read.me");
```

Note that this does not create a new connection to the resource, as an implicit UNC connection would if no NetUseAdd function was called.

- Implicit UNC connection. An implicit connection is made by calling to an OS/2 function and by passing the remote netname as part of a parameter. For example, the following call to DosOpen establishes an implicit UNC connection to the remote resource \\DEVELOP\\SRCDRV and opens FILE.1:

```
retcode = DosOpen ("\\\\develo\\srcdrv\\file.1", ...);
```

If the *ui1_password* field is set to NULL, the logon password is used to establish the first connection between a requester and a server. To establish a connection to a server on a domain where the user's password is different from that on the logon domain, the first NetUseAdd command must specify the password by which the user is defined on the server's domain. Subsequent connections to the same server will use the same password as that specified in the first NetUseAdd.

DOS Considerations

Under DOS, the NetUseAdd API cannot be used at the local requester to redirect communication serial devices.

Use Data Structures

The data structures used by the NetUseAdd, NetUseEnum, and NetUseGetInfo functions are described immediately following the syntax descriptions for each function.

Use Level 0


```

struct use_info_0 {
    unsigned char          ui0_local[DEVLEN+1];
    unsigned char          ui0_pad_1;
    unsigned char LSFAR *  ui0_remote;
};

```

where:

- *ui0_local* is an ASCIIZ string specifying the local device name (such as E:, LPT1, or COM1) being redirected to the shared resource.
- *ui0_pad_1* word-aligns the data structure component.
- *ui0_remote* points to an ASCIIZ string containing the netname of the remote resource being accessed. The string must be in the form *\\servername\netname*.

Use Level 1

```

struct use_info_1 {
    unsigned char          uil_local[DEVLEN+1];
    unsigned char          uil_pad_1;
    unsigned char LSFAR *  LSPTR uil_remote;
    unsigned char LSFAR *  LSPTR uil_password;
    unsigned short         uil_status;
    short                 uil_asg_type;
    unsigned short         uil_refcount;
    unsigned short         uil_usecount;
};

```

where:

- The first two fields in this data structure are identical to those in the previous level.
- *uil_remote* points to an ASCIIZ string specifying the UNC name of the remote resource being accessed. The string must be in the form *\\servername\netname*.
- *uil_password* points to an ASCIIZ string containing the password needed to establish a session between a specific requester and server. This field is used only by NetUseAdd.
- *uil_status* specifies the status of the connection. The following values for *uil_status* are defined in the USE.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
USE_OK	0	Connection valid.
USE_PAUSED	1	Paused by local requester.
USE_SESSLOST	2	Session removed.
USE_DISCONN	2	Connection disconnected.
USE_NETERR	3	Network error.
USE_CONN	4	Connection being made.
USE_RECONN	5	Reconnecting.

- *ui1_asg_type* specifies the type of remote resource being accessed. The following types of resources are defined in USE.H:

SYMBOLIC CONSTANT	VALUE	MEANING
USE_WILDCARD	-1	Matches the type of the share of the server. (Wildcards are used only when <i>uil_local</i> is a null string.)
USE_DISKDEV	0	Disk device.
USE_SPOOLDEV	1	Spooled printer.
USE_CHARDEV	2	Serial device.
USE_IPC	3	Interprocess communication (IPC).

To establish a connection with a null local device, set the *asg_type* field to USE_WILDCARD (-1). To establish a connection that maps a local device to the resource, use one of the other four values defined in the previous table.

- *ui1_refcount* indicates the number of files, directories, and other processes that are open on the remote resource.
- *ui1_usecount* indicates the number of explicit connections (redirection of a local device name) or implicit UNC connections (redirection of a null local device name) that are established with the resource.

If both an explicit and an implicit connection exist between a requester and a resource, the *usecount* of the server is 1 and the *usecount* of the requester is 2.

NetUseAdd or Net32UseAdd

NetUseAdd or Net32UseAdd

The NetUseAdd API establishes a connection between a local or null device name and a shared resource by redirecting the local or null (UNC) device name to the shared resource.

Restrictions

This API can be called from DLS and OS/2 workstations, but DOS does not support COM x serial devices when invoking this API locally. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <use.h>

NetUseAdd(pszServername, sLevel,
          buf, usBuflen);    /* 16 bit */

Net32UseAdd(pszServername, ulLevel,
            buf, ulBuflen);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in Use Level 1.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_BAD_DEV_TYPE	66	This network device type is incorrect.
ERROR_BAD_NET_NAME	67	This network name cannot be found.
ERROR_ALREADY_ASSIGNED	85	Duplicate redirection.
ERROR_INVALID_PASSWORD	86	The specified password is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_UseNotFound	2250	The connection cannot be found.
NERR_BadAsgType	2251	This asg_type is not valid.
NERR_DeviceIsShared	2252	This device is already being shared.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_LocalDrive	2405	The drive letter is in use locally.

Other codes could be returned from the following functions:

- DosFsRamSemRequest
- DosFsAttach
- DosFsCtl
- DosGetShrSeg

- DosSemRequest

Remarks

A local device name can be redirected to only one shared resource at a time. To establish a new direction for a redirected device, an application first must delete the existing connection by calling the NetUseDel function.

A COM or LPT device can be redirected even if it already is open. Redirecting the device has no effect on the handle of the device if it already is open. Subsequent openings, however, return a handle to the remote device to which the local device was redirected.

When a server is running user-level or share-level security, the *ui1_password* needed to access a shared resource can be provided in the following ways:

- A string that is not NULL specifies the password.
- A null pointer forces the OS/2 LAN Server software to use the same password given to the logon function.
- A null string indicates that no password is provided.

NetUseAdd ignores the *ui1_status*, *ui1_refcount*, and *ui1_usecount* components in the *use_info_1* data structure.

If a call to NetUseAdd duplicates an existing connection, the *usecount* component of the *use_info_1* data structure is incremented and the function succeeds, returning NERR_Success.

DOS users at a local requester cannot redirect COM \times serial devices with the NetUseAdd API.

Related Information

For information about:

- Disconnecting a device from a shared resource, see [Use - NetUseDel](#).
- Listing all devices redirected to a shared resource, see [Use - NetUseEnum](#).

NetUseDel or Net32UseDel

NetUseDel or Net32UseDel

The NetUseDel API ends a connection between a local or UNC device name and a shared resource.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <use.h>

NetUseDel(pszServername, netname,
         forceflag);    /* 16 bit */

Net32UseDel(pszServername, netname,
           forceflag);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

netname (const unsigned char LSFAR *) points to an ASCIIZ string that contains the following information:

- If the connection is a device name connection, the name specified must be the name of the local device.
- If the connection is a UNC connection (either implicit or explicit), the name must be the UNC name.

forceflag (16-bit unsigned short or 32-bit unsigned long) specifies three types of disconnection. The following values are defined in the USE.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
USE_NOFORCE	0	Maintains the connection in a dormant state, decrementing usecount. A dormant connection can be activated quickly as soon as resources are needed, improving system performance.
USE_FORCE	1	Connection is removed only if no more files, directories, or drive is opened. usecount is decremented for a local device name connection. usecount is forced to 0 for a UNC connection.
USE_LOTS_OF_FORCE	2	All files, directories, and drive connections are forced closed.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_INVALID_DRIVE	15	The specified drive is not valid.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_BAD_DEV_TYPE	66	This network device type is incorrect.
ERROR_BAD_NET_NAME	67	This network name cannot be found.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.

NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_OpenFiles	2401	There are open files on the connection.
NERR_DevInUse	2404	The device is being accessed by an active process.

Other codes could be returned from the following functions:

- DosFsAttach
- DosFsCtl

Related Information

For information about:

- Listing all local device names redirected to a shared resource, see [Use - NetUseEnum](#).
- Redirecting a local device name to a shared resource, see [Use - NetUseAdd](#).

NetUseEnum or Net32UseEnum

NetUseEnum or Net32UseEnum

The NetUseEnum API lists all current connections between the local computer and resources on a remote server.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <use.h>

NetUseEnum(pszServername, sLevel, buf, usBuflen,
          pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

Net32UseEnum(pszServername, ulLevel, buf, ulBuflen,
            pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Use Level 0](#) and [Use Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosDevIOCtl

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

Related Information

For information about:

- Listing the shared resources of a server, see [Share - NetShareEnum](#).
- Retrieving the status of a local device name, see [Use - NetUseGetInfo](#).

NetUseGetInfo or Net32UseGetInfo

NetUseGetInfo or Net32UseGetInfo

The NetUseGetInfo API retrieves information about a connection to a shared resource.

Restrictions

This API can be called from DLS and OS/2 workstations. Administrator authority is required to call this API remotely. However, no access authority is required for local execution.

Syntax

```
#include <netcons.h>
#include <use.h>

NetUseGetInfo(pszServername, netname, sLevel,
              buf, usBuflen, pusBytesAvail);    /* 16 bit */

Net32UseGetInfo(pszServername, netname, ulLevel,
                buf, ulBuflen, pulBytesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- netname* (const unsigned char LSFAR *) points to an ASCIIZ string that contains the following information:
- If the connection is a device name connection, the name specified must be the name of the local device.
 - If the connection is a UNC connection (either implicit or explicit), the name must be the UNC name.
- sLevel*/ or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [Use Level 0](#) and Use Level 1.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.

NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_UseNotFound	2250	The connection cannot be found.
NERR_InvalidComputer	2351	The specified computer name is not valid.

Other codes could be returned from the following functions:

- DosFsCtl
- DosDevIOCtl

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

The NetUseGetInfo function returns the *ui1_password* component as a null pointer, so users or applications cannot determine the password of another user or application.

Related Information

For information about:

- Listing all connections to a shared resource, see [Use - NetUseEnum](#).
- Listing the shared resources of a server, see [Share - NetShareEnum](#).

User Category

This category includes the following APIs:

[User - NetLogonEnum](#)
[User - NetUserAdd](#)
[User - NetUserDCDBInit](#)
[User - NetUserDel](#)
[User - NetUserEnum](#)
[User - NetUserGetAppSel](#)
[User - NetUserGetGroups](#)
[User - NetUserGetInfo](#)
[User - NetUserGetLogonAsn](#)
[User - NetUserModalsGet](#)
[User - NetUserModalsSet](#)
[User - NetUserPasswordSet](#)
[User - NetUserSetAppSel](#)
[User - NetUserSetGroups](#)
[User - NetUserSetInfo](#)
[User - NetUserSetLogonAsn](#)
[User - NetUserValidate2](#)

User APIs control a user's account in the user accounts subsystem (UAS) database for LAN Server and OS/2 Warp Server. The following APIs are used with the DCDB.H and NETCONS.H header files:

- NetUserDCDBInit
- NetUserGetAppSel

- NetUserGetLogonAsn
- NetUserSetAppSel
- NetUserSetLogonAsn

Each user or application that accesses the resources must have a user account in the system. The system uses this account to verify that the user or application has permission to connect to any shared resource. A user's account is set up by calling the NetUserAdd API.

There are three types of information in a user's account:

- That which can be set only by the administrator
- That which can be set by the owner of the account
- That which can be set only by the system

The following user account information can be set only by Administrator authority:

Account name	The name used to log on (for example, JSMITH).
Full name field	The full name of the user (for example, John Smith, Jr.). This field can be up to MAXCOMMENTSZ + 1 bytes long. A null pointer is treated as a null string.
Comment	The comment associated with an account (for example, Vice President). It can contain any text and can be up to MAXCOMMENTSZ + 1 bytes.
Home directory	The network directory (absolute, UNC, or local path) for the user's files. The field length can be up to PATHLEN bytes.
Account disabled	When set, no one can log on using this account.
Privilege level	Affects the default rights of an account to access resources and to call APIs remotely. An account can have one of three levels of privilege: GUEST, USER, or ADMIN. GUEST is not valid in DSS
Logon script flag	Must be TRUE for OS/2 LAN Server and DSS.
Logon script path	The path name of the user's logon script. The logon script must be a path relative to the NETLOGON service SCRIPTS path. This field can contain up to PATHLEN bytes, and a null pointer is treated as a null string. The extension can be PRO, CMD (or BAT), EXE, or COM. If no extension is specified, CMD (or BAT) is used by default.
Account expiration date	The date after which logon is disabled.
List of authorized requesters	The workstations from which the account can log on. The list may include 0-8 workstations. Zero workstations means ALL.
Logon hours	One hour granularity, hours specified for each day of the week. A null pointer means ALL for NetUserAdd calls and means DONT_CHANGE for NetUserSetInfo calls.
Home directory required	Indicates that the Home Directory field must not be NULL.
Password not required	Permits the account to have a null password, even if the minimum password length for the system is greater than 0. This option is not supported in DSS.
Maximum storage allotment	Specifies the limit in KB on the home directory.
User cannot change password	Restricts users from changing the password on their accounts. This option is not supported in DSS.
Logon server	Points to an ASCII string containing the name of the preferred server, which validates user logon requests for this user. The server name should be preceded by a double backslash (\\) and should be the name of a domain controller or backup server on the domain. A server name of an asterisk (*) indicates that the logon request can be handled by any domain controller or backup server on the domain. A null string indicates that the domain controller is the preferred logon server.
DCE Primary Group	Allows you to specify the distributed computing environment primary group. The default is "none".
DCE Policy Group	Allows you to specify the distributed computing environment policy group or organization. The default is "none".
Password Strength Server	Ensures that newly created or modified passwords follow specified rules of password composition. The DSS

password strength server (PSS) is "lspwssd".

The following fields can be set by a user (or by an administrator) only for the user's own account:

Encrypted password	When you create a password, it automatically becomes encrypted.
User comment	Can contain up to MAXCOMMENTSZ + 1 bytes long. A null pointer is treated as a null string.
Parms	Points to an ASCIIZ string that is set aside for use by applications. It can have as many as MAXCOMMENTSZ+1 characters.
Country code	The OS/2 country code for the language choice of the user. This is used by the OS/2 LAN Server software to generate messages in the appropriate language whenever possible.
Code page	The code page of the OS/2 program for national language support.

Default resource domain

An ASCIIZ string containing the name of the user's default resource domain.

The following fields are set automatically by the accounts system and are not set directly by an administrator or user:

Last logon	The time and date when the last logon occurred. A value of 0 means unknown.
Last logoff	The time and date the last logoff occurred. A value of 0 means unknown.
Bad password count	The number of attempts to validate a bad password. A value of -1 means unknown.
Number of logons	The number of instances of logons to the account. A value of -1 means unknown.

These fields are not used by DSS.

When a user or application requests access to any shared resource, LAN Server checks to see whether there is a UAS account granting the proper access authority to that user or application. To make this easy, first call the NetUserGetInfo API with a specified user ID and password. If an account is found, the API returns whatever information is available at that level of data structure. If an account is not found, you can examine the returned flags to determine what must be done to establish an account. *

The OS/2 LAN Server software then checks the user's privilege level. Depending on the parameter, the request to access a resource is accepted immediately, if the user has been given administrative privileges. Otherwise, processing continues. *

Administrative privileges grant the broadest access to the domain, giving permission to run all administrative functions and complete access to the shared and nonshared resources of a server.

If the user does not have administrative privileges, the LAN Server software checks the resource's access permission record to see whether the user has the proper permissions to use that particular resource. (See [Access Permission Category](#).)

Each time an account that has either administrative or user privileges is established with the NetUserAdd API, LAN Server automatically adds the new account to either the group ADMINS or the group USERS. At this time, the user inherits all permissions assigned to that special group.

An application has two ways to change a user's current privileges: (1) by calling the NetUserSetInfo API, or (2) by changing that user's group accounts (see [Group Category](#)). Individually assigned user permissions take precedence over group permission assignments. An application can verify to which groups a user belongs by calling the NetUserGetGroups API. This API returns a list of group names.

To find out how to change individual permission settings for members of one of the special groups, see [Access Permission Category](#).

When a user account is no longer needed, call the NetUserDel API to eliminate the account from the system. Once the account is removed, the user no longer can access the system.

If an account does not have a password, any password (or none) is treated as a match for account validation. *

To set password policy globally within the account system database, you can use user accounts subsystem (UAS) modals. None of the system-wide password characteristics are enforced for accounts that do not require passwords. *

Enter passwords in uppercase characters. The password of a user account is case-sensitive at the API level, and lowercase characters in a password make the user ID unavailable at the command-line interface and through the User Profile Management (UPM) interface.

The user's password is confidential and is not returned when the NetUserEnum or NetUserGetInfo API is called; a string of spaces is substituted for any password that is requested. The password is assigned initially when NetUserAdd is called. Any user or application can use NetUserPasswordSet to change a password, if the current password can be supplied. To verify an existing user account with a specified password, call NetUserValidate2. To set the password and other components of a user account, use the NetUserSetInfo API.

To give a user access to network resources at logon, call the NetUserSetLogonAsn API. You can get information about a user's logon assignments by calling the NetUserGetLogonAsn API.

To add network applications to their desktop applications folders, users can call the NetUserSetAppSel API. Users can get lists of the

applications that appear in their desktop applications folders by calling the NetUserGetAppSel API.

* This information does not apply to DSS. This specific information is part of LAN Server and OS/2 Warp Server.

DOS Considerations

LAN Server and OS/2 Warp Server Only

The NetUserPasswordSet API controls a user's password account in a domain. All of the APIs in this category (except NetUserValidate2) can be called from DLS workstations, but only to remote LAN Servers. NetUserValidate2 is not supported for DLS workstations.

There are five types of data structures in the User category: user account, user modal, user validation, application selector, and logon assignment.

A DLS peer workstation maintains its own user account database and access control lists locally. The peer workstation's user account is never synchronized with any other user accounts database, like OS/2 LAN Server domain's UAS.

A DLS peer workstation cannot handle logon requests and therefore, cannot be a logon server.

Differences in OS/2 LAN Server Corresponding API

Some fields of data structures are not supported by the user level security functions for DLS although they are supported by OS/2 LAN Server. They are:

Table 4. Structure user_info_1:

MEMBER NAME	DIFFERENCES IN OS/2 LAN SERVER DEFINITION
char FAR *usril_homedir	The value must be NULL. DOS Peer does not prepare home directory function.
ushort usril_flags	The bit of UF_SCRIPT(0x01, logon script bit) and UF_HOMEDIR_REQUIRED(0x08) must be zero.
char FAR *usril_script_path	Must be NULL.

Table 5. Structure user_info_2:

MEMBER NAME	DIFFERENCES IN OS/2 LAN SERVER DEFINITION
char FAR *usri2_parms	Must be zero.
long usri2_last_logon	A value of 0 is always set.
long usri2_last_logoff	A value of 0 is always set.

ulong usri2_max_storage	DLS does not use this field. Upon return, a value of -1 is set.
ushort usri2_units_per_week	DLS does not use this field.
uchar FAR *usri2_logon_hours	Must be NULL.
ushort usri2_bad_pw_count	The value 0xFFFF is always returned.
ushort usri2_num_logons	The value 0xFFFF is always returned.
char FAR *usri2_logon_server	Must be NULL.

User Account Data Structures

The data structures described here represent levels of detail available for the functions that define or return individual user accounts. The NetUserAdd, NetUserEnum, NetUserGetInfo, and NetUserSetInfo APIs use *user_info_0*, *user_info_1*, *user_info_2*, *user_info_10*, or *user_info_11*, depending on whether their *level* parameter is 0, 1, 2, 10, 11 or 201.

User Account Information Level 0

```
struct user_info_0 {
    unsigned char  usri0_name[UNLEN+1];
};
```

where *usri0_name* specifies the name of the user.

User Account Information Level 1

```
struct user_info_1 {
    unsigned char          usril_name[UNLEN+1];
    unsigned char          usril_pad_1;
    unsigned char          usril_password[ENCRYPTED_PWLEN];
    long                   usril_password_age;
    unsigned short         usril_priv;
    unsigned char LSFAR * LSPTR usril_home_dir;
    unsigned char LSFAR * LSPTR usril_comment;
    unsigned short         usril_flags;
    unsigned char LSFAR * LSPTR usril_script_path;
};
```

where:

- The first field in this data structure is identical to that in the previous level.
- *usri1_pad_1* word-aligns the data structure component.

- *usr1_password* is the password of *usr1_name*. The NetUserEnum and NetUserGetInfo APIs return a string of spaces to maintain password security. The string can be NULL. It can contain as many as PWLEN bytes, defined in NETCONS.H. The password should be uppercase.
- *usr1_password_age* indicates the number of seconds that have passed since *usr1_password* last changed. NetUserAdd and NetUserSetInfo calls ignore this element.

To allow room for the encrypted version of the password to be transmitted over the network, the *usr1_password* field is longer than PWLEN bytes. When setting a user's password, check its length against PWLEN, not against ENCRYPTED_PWLEN.

- *usr1_priv* is one of three values indicating the level of privilege assigned *usr1_name*. The ACCESS.H header file defines these values as follows:

SYMBOLIC CONSTANT	VALUE	PRIVILEGE
USER_PRIV_GUEST	0	Guest (not supported in DSS)
USER_PRIV_USER	1	User
USER_PRIV_ADMIN	2	Administrator

- *usr1_home_dir* points to an ASCIIZ string specifying the path to the user's home directory. (If this field or the string to which it points is null, no home directory is assigned for that user.) The home directory string must take one of the following forms:

- A form that defines the drive of the user that will be assigned to the home directory. It can have as many as PATHLEN bytes.

x: \machineID\y\$\pathname

where:

- *x* is the drive letter to be assigned.
- *machineID* is the name of the server that holds the home directory.
- *y* is the drive letter on the server where the home directory exists.
- *pathname* is the remaining path to the directory.

- A form that defines a first-available drive specification for the name directory:

\machineID\y\$\pathname

where *machineID*, *y*, and *pathname* are defined as in the preceding form.

To define a home directory at the root of a server's drive, omit *pathname* from either of the forms previously described.

Note: You also must create an access control profile to give users access to the home directory. See [Access Permission Category](#) for those APIs.

- *usr1_comment* points to an optional ASCIIZ string containing an optional comment or remark about the user. The string can be NULL, or can have as many as MAXCOMMENTSZ bytes before the terminating NULL character.
- *usr1_flags* determines whether a logon script is to be run and whether the user's account is enabled. The ACCESS.H header file defines the following possible values:

SYMBOLIC CONSTANT	BIT MASK	MEANING
UF_SCRIPT	0x1	Must be 1. Logon script enabled.
UF_ACCOUNT_DISABLE	0x2	If 1, user's account disabled.
UF_DELETE_PROHIBITED	0x4	If 1, user's account cannot be deleted.
UF_HOMEDIR_REQUIRED	0x8	If 1, home directory required.
UF_PASSWD_NOTREQD	0x20	If 1, password not required. (not supported in DSS)

UF_PASSWD_CANT_CHANGE	0x40	If 1, user cannot change password. (not supported in DSS)
-----------------------	------	--

- *usri1_script_path* points to an ASCIIZ string indicating the path name of the user's logon script (CMD, EXE, BAT or PRO file). If a file extension is not used, the file is assumed to have a CMD or BAT extension.

The script path must be specified relative to the Netlogon service SCRIPTS path. It can have as many as PATHLEN bytes before the terminating NULL character. The constant PATHLEN is defined in the NETCONS.H header file. A null string indicates that no logon script exists.

User Account Information Level 2

Information level 2 accommodates additional information for the UAS database. It is an extension of *user_info_1*.

```
struct user_info_2 {
    unsigned char    usri2_name[UNLEN+1];
    unsigned char    usri2_pad_1;
    unsigned char    usri2_password[ENCRYPTED_PWLEN];
    long             usri2_password_age;
    unsigned short   usri2_priv;
    unsigned char LSFAR * LSPTR usri2_home_dir;
    unsigned char LSFAR * LSPTR usri2_comment;
    unsigned short   usri2_flags;
    unsigned char LSFAR * LSPTR usri2_script_path;
    unsigned long    usri2_auth_flags;
    unsigned char LSFAR * LSPTR usri2_full_name;
    unsigned char LSFAR * LSPTR usri2_usr_comment;
    unsigned char LSFAR * LSPTR usri2_parms;
    unsigned char LSFAR * LSPTR usri2_workstations;
    long             usri2_last_logon;
    long             usri2_last_logoff;
    long             usri2_acct_expires;
    unsigned long    usri2_max_storage;
    unsigned short   usri2_units_per_week;
    unsigned char LSFAR * LSPTR usri2_logon_hours;
    unsigned short   usri2_bad_pw_count;
    unsigned short   usri2_num_logons;
    unsigned char LSFAR * LSPTR usri2_logon_server;
    unsigned short   usri2_country_code;
    unsigned short   usri2_code_page;
};
```

where:

- The first nine fields in this data structure are identical to those in the previous level.
- *usri2_auth_flags* grants operator privileges (accounts, print, comm, server) to users. The ACCESS.H header file defines the following possible values:

SYMBOLIC CONSTANT	BITS	MEANING
AF_OP_PRINT	0	If 1, print operator privilege is enabled.
AF_OP_COMM	1	If 1, comm operator privilege is enabled.
AF_OP_SERVER	2	If 1, server operator privilege is enabled.
AF_OP_ACCOUNTS	3	If 1, accounts operator privilege is enabled. (not supported in DSS)
n/a	4-31	Reserved; must be 0.

- *usr2_full_name* points to an ASCIIZ string containing the full name of the user. The string can be NULL or can have as many as MAXCOMMENTSZ + 1 bytes.
- *usr2_usr_comment* points to an ASCIIZ string that is a user-settable field. The string can be NULL or it can have as many as MAXCOMMENTSZ + 1 bytes.
- *usr2_parms* points to an ASCIIZ string that is set aside for use by applications. The string can be NULL or it can have as many as MAXCOMMENTSZ + 1 bytes.
- *usr2_workstations* points to an ASCIIZ string containing a list of requesters from which a user is permitted to log on. A null string means all requesters are allowed. (To disallow logon, the account disabled flag must be set.) Up to 8 requesters can be specified, where the names are separated by spaces (0x20). The list of requesters can include IBM NetBIOS permanent names, which are listed as machine IDs, consisting of 12 hexadecimal characters. IBM NetBIOS permanent names are entered in the requester as follows:

16DF.02AC.7DE9

- *usr2_last_logon* is the time and date (seconds since 1 January 1970) when the last logon occurred. A value of 0 means unknown. This field can be set only by the system; it is ignored in NetUserAdd and NetUserSetInfo calls. DSS always returns 0.
- *usr2_last_logoff* is the time and date (seconds since 1 January 1970) when the last logoff occurred. A value of 0 means unknown. This field can be set only by the system; it is ignored in NetUserAdd and NetUserSetInfo calls. DSS always returns 0.
- *usr2_acct_expires* is the time and date (seconds since 1 January 1970) when the account will expire. An expired account is the equivalent of a disabled account. An entry of 0xFFFFFFFF or (-1) means there is no expiration date.
- *usr2_max_storage* is the maximum storage allotted for the home directory. The units are kilobytes (KB). An entry of 0xFFFFFFFF or (-1) means unlimited storage. The field is not enforced by the system but is available to utilities such as CHKSTOR. Those utilities generate reports and send alerts when they determine that users have exceeded their maximum storage.
- *usr2_units_per_week* is the number of equal-length time units into which the week is divided. This value is used to compute the length of the bit string in *logon_hours*. It must be UNITS_PER_WEEK (168). This field is ignored by NetUserAdd and NetUserSetInfo calls.
- *usr2_logon_hours* points to a 21-byte (168 bits) map, in which each bit represents a unique hour in a week. The first bit (bit 0, word 0) is Sunday, 0:00 to 0:59. Bit 1, word 0 is Sunday, 1:00 to 1:59, and so on. If a bit is set in this bitmap, logon is allowed. If a bit is cleared, logon is not allowed.

Note: A null pointer in this field for a NetUserAdd call permits access at all times. A null pointer in this field for a NetUserSetInfo call means that no change is made.

- *usr2_bad_pw_count* is the number of attempts to validate a bad password. A value of -1 means unknown. This field can be set only by the system and is ignored in NetUserAdd and NetUserSetInfo calls. DSS always returns -1.
- *usr2_num_logons* is the number of instances of logons to the account. A value of -1 means unknown. DSS always returns -1. This field can be set only by the system and is ignored by NetUserAdd and NetUserSetInfo.
- *usr2_logon_server* points to an ASCIIZ string that contains the name of the preferred server that validates user logon requests for this user. The server name should be preceded by a double backslash (\\) and should be the name of a domain controller or backup server on the domain. A server name of an asterisk (*) indicates that the logon request can be handled by any domain controller or backup on the domain. A null string indicates that the domain controller is the logon server.
- *usr2_country_code* specifies the OS/2 country code for the user's language choice. This is used by the LAN Server software to generate messages in the appropriate language whenever possible.
- *usr2_code_page* is the OS/2 code page for the language choice of the user.

User Account Information Level 10

This data structure allow users to retrieve information about other users.

```
struct user_info_10 {
    unsigned char          usr10_name[UNLEN+1];
```



```

    unsigned char          usri10_pad_1;
    unsigned char LSFAR * LSPTR usri10_comment;
    unsigned char LSFAR * LSPTR usri10_usr_comment;
    unsigned char LSFAR * LSPTR usri10_full_name;
};

```

where each field in this level is identical to the corresponding field in the previous level.

User Account Information Level 11

This data structure allows users to retrieve a higher degree information about themselves.

```

struct user_info_11 {
    unsigned char          usri11_name[UNLEN+1];
    unsigned char          usri11_pad_1;
    unsigned char LSFAR * LSPTR usri11_comment;
    unsigned char LSFAR * LSPTR usri11_usr_comment;
    unsigned char LSFAR * LSPTR usri11_full_name;
    unsigned short         usri11_priv;
    unsigned long          usri11_auth_flags;
    long                  usri11_password_age;
    unsigned char LSFAR * LSPTR usri11_home_dir;
    unsigned char LSFAR * LSPTR usri11_parms;
    long                  usri11_last_logon;
    long                  usri11_last_logoff;
    unsigned short         usri11_bad_pw_count;
    unsigned short         usri11_num_logons;
    unsigned char LSFAR * LSPTR usri11_logon_server;
    unsigned short         usri11_country_code;
    unsigned char LSFAR * LSPTR usri11_workstations;
    unsigned long          usri11_max_storage;
    unsigned short         usri11_units_per_week;
    unsigned char LSFAR * LSPTR usri11_logon_hours;
    unsigned short         usri11_code_page;
};

```

where all the fields in this level are identical to those in the previous levels.

User Account Information Level 201

This data structure provides support for DSS-specific user attributes.

```

struct user_info_201 {
    unsigned char          usri201_name[UNLEN+1];
    unsigned char          usri201_pad_1[3];
    unsigned char LSFAR * LSPTR usri201_full_name;
    unsigned char LSFAR * LSPTR usri201_comment;
    unsigned char LSFAR * LSPTR usri201_usr_comment;
    unsigned char          usri201_password[ENCRYPTED_PWLEN];
    long                  usri201_password_age;
    unsigned long          usri201_priv;
    unsigned char LSFAR * LSPTR usri201_home_dir;
    unsigned long          usri201_flags;
    unsigned char LSFAR * LSPTR usri201_script_path;
    unsigned long          usri201_auth_flags;
    unsigned char LSFAR * LSPTR usri201_parms;
    unsigned char LSFAR * LSPTR usri201_workstations;
    long                  usri201_acct_expires;
    unsigned long          usri201_max_storage;
    unsigned char LSFAR * LSPTR usri201_logon_hours;
    unsigned char LSFAR * LSPTR usri201_logon_server;
    unsigned long          usri201_country_code;
    unsigned long          usri201_code_page;
    unsigned char LSFAR * LSPTR usri201_default_realm;
};

```

```

unsigned char LSFAR * LSPTR    usri201_dce_primary_group;
unsigned char LSFAR * LSPTR    usri201_dce_policy_group;
unsigned char LSFAR * LSPTR    usri201_pwstrength_server
} user_info_201_t;

```

where:

- All fields are identical to those defined in level 2 with the following exceptions:
- All unsigned short fields are now unsigned long.
- *usri201_default_realm* points to an ASCIIZ string containing the name of the user's default resource domain.
- *usri201_dce_primary_group* points to an ASCIIZ string containing the name of the user's primary group.
- *usri201_dce_policy_group* points to an ASCIIZ string containing the name of the user's policy group (also known as organization).
- *usri201_pwstrength_server* points to an ASCIIZ string containing the name of the password strength server that will perform composition checks on the user's password. This may not be NULL.

User Modals Data Structures

To control global modals, use the following data structures.

User Modals Level 0

```

struct user_modals_info_0 {
    unsigned short  usrmod0_min_passwd_len;
    unsigned long   usrmod0_max_passwd_age;
    unsigned long   usrmod0_min_passwd_age;
    unsigned long   usrmod0_force_logoff;
    unsigned short  usrmod0_password_hist_len;
    unsigned short  usrmod0_maxbadpw;
};

```

where:

- *usrmod0_min_passwd_len* is the minimum password length. The range of values is 0 to MAX_PASSWD_LEN.
- *usrmod0_max_passwd_age* is the maximum time (in seconds) since the password was last changed and for which the current password is valid. A value of 0xFFFFFFFF (TIMEQ_FOREVER) allows the password to be valid forever. The minimum value is 1 day.
- *usrmod0_min_passwd_age* is the minimum time (in seconds) since the password was last changed and before which the current password is allowed to be changed. A value of 0 means no delay is required between password updates.
- *usrmod0_force_logoff* is the length of time (in seconds) after the valid logon hours that the user should be forced off the network. If the value is TIMEQ_FOREVER (0xFFFFFFFF), the user is never forced off. If the value is 0, the user is forced off immediately. Any value between these also can be used.
- *usrmod0_password_hist_len* is the length of the password history (that is, the number of passwords in the history buffer that are scanned as opposed to the new password in a NetUserPasswordSet attempt). The new password must not match any of the entries scanned. The history is of encrypted passwords. It can be set from 0 to DEF_MAX_PWHIST (currently 8).
- *usrmod0_maxbadpw* restricts the number of bad password attempts. A value of 0 sets no limits. This parameter is not supported in DSS.

User Modals Level 1

```
struct user_modals_info_1 {
    unsigned short      usrmodl_role;
    unsigned char LSFAR * LSPTR  usrmodl_primary;
};
```

where:

- *usrmodl_role* is the role of this server under a single system image (SSI). It can be one of the following:

SYMBOLIC CONSTANT	VALUE	MEANING
UAS_ROLE_STANDALONE	0	Standalone server
UAS_ROLE_MEMBER	1	Member server in the domain
UAS_ROLE_BACKUP	2	Backup server in the domain
UAS_ROLE_PRIMARY	3	Primary server in the domain

Without SSI, this field always should be set to STANDALONE.

- *usrmodl_primary* is the name of the domain controller of the primary domain. It should match the primary domain name of the requester software on the local machine.

User Logon Data Structures

The following are the logon data structures.

User Logon Level 0

```
struct user_logon_info_0 {
    unsigned char      usrlog0_eff_name[UNLEN+1];
    unsigned char      usrlog0_pad_1;
};
```

where:

- *usrlog0_eff_name* is the UAS account name (user ID) submitted with the API call.
- *usrlog0_pad* word-aligns the data structure component.

User Logon Level 1

```
struct user_logon_info_1 {
    unsigned short      usrlog1_code;
    unsigned char       usrlog1_eff_name[UNLEN+1];
    unsigned char       usrlog1_pad_1;
    unsigned short      usrlog1_priv;
    unsigned long        usrlog1_auth_flags;
    unsigned short      usrlog1_num_logons;
    unsigned short      usrlog1_bad_pw_count;
    unsigned long        usrlog1_last_logon;
    unsigned long        usrlog1_last_logoff;
    unsigned long        usrlog1_logoff_time;
    unsigned long        usrlog1_kickoff_time;
    long                usrlog1_password_age;
    unsigned long        usrlog1_pw_can_change;
    unsigned long        usrlog1_pw_must_change;
    unsigned char LSFAR * LSPTR usrlog1_computer;
    unsigned char LSFAR * LSPTR usrlog1_domain;
    unsigned char LSFAR * LSPTR usrlog1_script_path;
    unsigned long        usrlog1_reserved1;
};
```

where:

- usrlog1_code* specifies a response from the API, according to the following values:

SYMBOLIC CONSTANT	VALUE	MEANING
VALIDATED_LOGON	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	The username and password do not correspond to an active account.
NERR_NonValidatedLogon	2217	The logon server could not validate the logon.
NERR_InvalidRequester	2240	The user is not allowed to log on from this requester.
NERR_InvalidLogonHours	2241	The user is not allowed to log on at this time.
NERR_PasswordExpired	2242	The password has expired.
- The second and third fields in the *user_logon_info_1* are identical to the corresponding fields in the previous level.
- usrlog1_priv* specifies the user's privilege level. It can be one of the following:

SYMBOLIC CONSTANT	VALUE	PRIVILEGE
USER_PRIV_GUEST	0	Guest
USER_PRIV_USER	1	User
USER_PRIV_ADMIN	2	Administrator
- usrlog1_auth_flags* grants operator rights (accounts, print, comm, and server) to users, both locally and remotely. See the chart about operator privileges at topic [User Account Information Level 2](#).
- usrlog1_num_logons* is the number of instances of logons to the account. A value of -1 means the number of logons is unknown.

- *usrlog1_bad_pw_count* is the number of attempts to validate a bad password. A value of -1 means the number is unknown.
- *usrlog1_last_logon* is the time and date (seconds since 1 January 1970) the user last logged on.
- *usrlog1_last_logoff* is the time and date (seconds since 1 January 1970) when the last logoff occurred. A value of 0 means the time is unknown.
- *usrlog1_logoff_time* is the time (in seconds) and date (since 1 January 1970) when the user should log off. A value of USER_NO_LOGOFF means the user never has to log off. The constant USER_NO_LOGOFF is defined in the ACCESS.H header file.
- *usrlog1_kickoff_time* is the time (in seconds) when the user will be logged off. A value of USER_NO_LOGOFF means the user will not be logged off by the system. The constant USER_NO_LOGOFF is defined in the ACCESS.H header file.
- *usrlog1_password_age* is the time (in seconds) since the password last was changed.
- *usrlog1_pw_can_change* is the time (in seconds) and date (since 1 January 1970) when the user can change his password. A value of TIMEQ_FOREVER, as defined in the ACCESS.H header file, means the user never can change the password.
- *usrlog1_pw_must_change* is the time (in seconds) and the date (since 1 January 1970) when the password must be changed.
- *usrlog1_computer* is the server that validated the user's logon.
- *usrlog1_domain* is the domain to which the user currently is logged on.
- *usrlog1_script_path* is the relative path to the logon script of an account.
- *usrlog1_reserved1* is reserved and cannot be used.

User Logon Level 2

```
struct user_logon_info_2 {
    unsigned char          usrlog2_eff_name[UNLEN+1];
    unsigned char          usrlog2_pad_1;
    unsigned char LSFAR * LSPTR usrlog2_computer;
    unsigned char LSFAR * LSPTR usrlog2_full_name;
    unsigned char LSFAR * LSPTR usrlog2_usrcomment;
    unsigned long          usrlog2_logon_time;
};
```

where:

- The first three fields in this data structure are identical to those in the previous level.
- *usrlog2_full_name* is the full name of the user.
- *usrlog2_usrcomment* points to an ASCIIZ string that is a user-settable field for user comments. The string can be NULL.
- *usrlog2_logon_time* is the time and date (seconds since 1 January 1970) when the user logged on. It can be NULL for local. A null pointer indicates that the workstation is ignored.

User Logon Request Level 1

```
struct user_logon_req_1 {
    unsigned char          usrreq1_name[UNLEN+1];
    unsigned char          usrreq1_pad_1;
    unsigned char          usrreq1_password[SESSION_PWLEN];
};
```

```

    unsigned char LSFAR * LSPTR    usrreq1_workstation;
};

```

where:

- *usrreq1_name* specifies the user or group ID being submitted for logon.
- *usrreq1_pad* word-aligns the data structure component.
- *usrreq1_password* contains the ASCII password to be submitted.

If the name passed in the *usrreq1_name* field has no password assigned to its account in the UAS database, any value in this password field is valid.

If the password fails to match, the bad password count field in the user's record is increased by 1. A successful match resets the field.

- *usrreq1_workstation* points to an ASCIIZ string containing the name of the requester from which the user is logging on. It can be NULL for local. A null pointer indicates that the workstation is ignored.

User Logoff Level 1

The following is the logoff data structure:

```

struct user_logoff_info_1 {
    unsigned short usrlogfl_code;
    unsigned long  usrlogfl_duration;
    unsigned short usrlogfl_num_logons;
};

```

where:

- *usrlogfl_code* is dependent on the calling API. It is specified separately for each API.
- *usrlogfl_duration* is the duration of a logon. It can be 0 for unknown.
- *usrlogfl_num_logons* is the number of logons by a user name. It can be -1 for unknown.

User Application Data Structures

User application data structures consist of a fixed-length header structure (*app_sel_info_1*), followed by 0 or more *app_sel_list* data structures. (An *app_sel_list* structure is returned for each application in query.)

Application Data Structure Header

```

struct app_sel_info_1 {
    unsigned long  asil_reserved;
    unsigned short asil_count;
};

```

where:

- *asi1_reserved* is reserved for future use. It must be 0.
- *asi1_count* indicates the number of *app_sel_list* structures that immediately follow the *app_sel_info_1* structure.

```
struct app_sel_info_200 {
    unsigned long    asi200_count;
} app_sel_info_200_t;
```

where *asi200_count* is identical to *asi1_count* except that it is now an unsigned long.

Application List

```
struct app_sel_list {
    unsigned char    asl_appname[APP_LEN + 1];
    unsigned char    asl_pad_1;
    unsigned short   asl_apptype;
    unsigned short   asl_reserved;
};
```

where:

- *asl_appname* is the name of an application that exists in the domain control database. In an application name, use neither an embedded blank nor the following characters defined in the DCDB.H header file:

"./\[\]: | < > + = ; , ? *

- *asl_pad_1* word-aligns the data structure component.
- *asl_apptype* is one of the following five values, defined in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_DOS_PUBLIC	0x01	Public DOS applications.
APP_OS2_PUBLIC	0x02	Public OS/2 applications.
APP_OS2_PRIVATE	0x04	Private OS/2 applications.
APP_DOS_PRIVATE	0x08	Private DOS applications.
APP_TYPE_UNKNOWN *1	0x040	The type of this application cannot be determined.

*1: Not valid for a NetUserSetAppSel API call.

If returned from NetUserGetAppSel, the definition for that application might have been deleted. In that case, use the NetUserSetAppSel API to delete the application from that user's list.

- *asl_reserved* is reserved for future use.

```
struct app_sel_list_200 {
    unsigned char    asl200_appname[APP_LEN + 1];
    unsigned char *   asl200_global_name;
    unsigned long     asl200_apptype;
};
```

where:

- *asl200_appname* is identical to *asl_appname*
- *asl200_global_name* points to an ASCIIZ string that contains the global name of the application. The string may be up to CDS_FNAME_MAX(1024) (defined in cdsclerk.h) characters.

- *asl200_apptype* is identical to *as1_apptype* except it is now an unsigned long.

Logon Assignment Data Structure

```
struct logon_asn_info_1 {
    unsigned long    lai1_reserved;
    unsigned short   lai1_count;
};
```

where:

- *lai1_reserved* is reserved for future use. It must be 0.
- *lai1_count* indicates the number of *logon_asn_list* structures that immediately follow the *logon_asn_info_1* structure.

```
struct logon_asn_info_200 {
    unsigned long    lai200_count;
}logon_asn_info_200_t;
```

where *lai200_count* indicates the number of *logon_asn_list_200* structures that immediately follow the *logon_asn_list_200* structure.

```
struct logon_asn_list {
    unsigned char     lai_alias[ALIAS_LEN + 1];
    unsigned char     lai_pad;
    unsigned short    lai_type;
    unsigned char     lai_device[DEVLEN + 1];
};
```

where:

- *lai_alias* specifies an existing alias for the resource that is to be assigned automatically to the user at logon time.
- *lai_pad* word-aligns the data structure component.
- *lai_type* is one of three values indicating the alias type. The DCDB.H header file defines these values as follows:

SYMBOLIC CONSTANT	VALUE	MEANING
ALIAS_TYPE_FILE	0x0001	Files alias
ALIAS_TYPE_PRINTER	0x0002	Printer alias
ALIAS_TYPE_SERIAL	0x0004	Serial device alias

- *lai_device* indicates the device that is assigned to the resource specified by *lai_alias* when the user logs on. Valid drive letters are A-Z. The drive letter can be followed by a colon (:), but a colon is not required. An asterisk (*) indicates that the system chooses the first available drive.

Valid print devices are LPT1-LPT9. Valid serial devices are LPT1-LPT9 and COM1-COM16. Print and serial devices should not be followed by a colon (that is, *COM1:* would be an invalid serial device). The first byte of *lai_device* can be a space character (0x20) or a NULL character (0x00). If the first byte is one of these characters, the corresponding alias is not assigned a device.

```
struct logon_asn_list_200 {
    unsigned char     lai200_alias[ALIAS_LEN + 1];
    unsigned char     lai200_global_name;
    unsigned long     lai200_type;
    unsigned char *   lai200_device;
}logon_asn_list_200_t;
```

where:

- *lal200_alias* is identical to *lal_alias*
- *lal200_global_name* points to an ASCIIZ string, that contains the global name of an existing alias for the resource that is to be assigned automatically to the user at logon time. The string may be up to CDS_FNAME_MAX(1024) characters.
- *lal200_type* is identical to *lal_type* except that it is now an unsigned long.
- *lal200_device* points to an ASCIIZ string that contains the device that is assigned to the resource specified by *lal200_global_name* when the user logs on.

Related Information

For information about:

- Logon scripts, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Share resource access permissions, see [Share Category](#).

NetLogonEnum or Net32LogonEnum

NetLogonEnum or Net32LogonEnum

The NetLogonEnum API supplies information about logged-on users.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

Syntax

```
#include <netcons.h>
#include <access.h>

NetLogonEnum(pszServername, sLevel, buf,
             usBuflen, pusEntriesReturned, pusEntriesAvail); /* 16 bit */

Net32LogonEnum(pszServername, ulLevel, buf,
               ulBuflen, pulEntriesReturned, pulEntriesAvail); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 2, specifying which data structure to use, as described in [User Logon Level 0](#) and [User Logon Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_NetLogonNotStarted	2455	The Netlogon service has not been started.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosFsCtl
- DosGetShrSeg

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

NetUserAdd or Net32UserAdd

NetUserAdd or Net32UserAdd

The NetUserAdd API establishes an account for a user in the user accounts subsystem (UAS) database and assigns a password and privilege level. NetUserAdd allows adding a new user ID for the additional server.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level. You cannot add a user with GUEST privilege. Doing so would result in an ERROR_INVALID_PARAMETER being returned. Users and groups are allowed to have the same name. Privilege and authority flags are ignored when *pszTarget* resolves to a cell. This API is only valid from a DSS client/server when targeted at DSS.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetUserAdd(pszTarget, usLevel,
           buf, usBuflen);      /* 16 bit */

Net32UserAdd(pszTarget, ulLevel,
             buf, ulBuflen, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszTarget</i>	is a place holder you replace depending on whether you are connecting to an OS/2 LAN Server or DSS. The following are examples of each.
<i>null or lservername</i>	user added to cell and server's resource domain. or OS/2 LAN Server or OS/2 Warp Server.
<i>//resdomname</i>	user added to cell and resource domain. cell.
<i>//resdomname @cellname</i>	user added to the specified cell and the resource domain of the specified cellname.
<i>./.: or /... /cellname</i>	adds users to a cell.
<i>sLevel or ulLevel</i>	(16-bit short or 32-bit unsigned long) can be 1,2, or 201 specifying which data structure to use, as described in User Account Information Level 1

and [User Account Information Level 2](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PASSWORD	86	The specified password is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_GroupExists	2223	The group name is already in use.
NERR_UserExists	2224	The user account already exists.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.

NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_PasswordTooShort	2245	The password is shorter than required.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_OrgNotFound	7510	The specified organization/policy group does not exists.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_UserExistsInCell	7514	The user already exists in the cell..
NERR_PrimaryGroupNotFound	7536	The specified primary group does not exist.
NERR_EncryptedPassword	7554	While attempting to set the user encrypted password, an error occurred.
NERR_PasswordHistory	7555	While attempting to update the user password history, an error occurred.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Remarks

For calls at level 1, the additional fields in the *user_info_2* data structures are initialized to the following default values. (You can set these fields to the desired values with subsequent NetUserSetInfo calls.)

usri2_full_name	<i>usri2_name</i>
usri2_usr_comment	None (null string)
usri2_parms	None (null string)
usri2_workstations	All (null string)
usri2_acct_expires	Never (0xFFFFFFFF)
usri2_max_storage	Unlimited (0xFFFFFFFF)
usri2_logon_hours[]	Logon allowed anytime (each element 0xFF; all bits set to 1)
usri2_logon_server	Domain controller (null string)

usr2_country_code	Current country code on the server
usr2_code_page	Current code page on the server
usr2_auth_flags	None (0)

usr2_password_age	Sets the base time equal to the time of creation
reserved	All reserved fields

For calls at levels 1 and 2, the additional fields in the *user_info_201* data structure are initialized to the following defaults values:

Related Information

- Listing user accounts on a server, see [User - NetUserEnum](#).
- Logon scripts, see the *LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*.
- Changing a user account, see [User - NetUserSetInfo](#).
- Changing a user's group memberships, see [Group Category](#).
- Removing a user account, see [User - NetUserDel](#).

NetUserDCDBInit or Net32UserDCDBInit

Restrictions

- The calling process has Administrator authority.
- The calling process has accounts operator privilege.
- The calling process has User authority and is initializing the domain control database for the logged-on user.

```
#include <netcons.h>
#include <dcdb.h>

NetUserDCDBInit(pszServername, pszUserID,
                reserved);                                /* 16 bit */

Net32UserDCDBInit(pszServername, pszUserID,
                 reserved, pszReservedX);              /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszServername</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the domain controller in which <i>pszUserID</i> is defined. It can be null only if the calling application is running on the domain controller.
<i>reserved</i>	(unsigned long) must be 0.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_DATA	13	The specified data is not valid.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr
- NetAccessAdd
- NetAccessGetInfo
- NetAccessSetInfo
- DosQFileMode
- DosOpen2
- DosWrite

Remarks

You can assume a user's domain control database (DCDB) area has not been initialized if the NetUserGetLogonAsn API is called and the ERROR_PATH_NOT_FOUND code is returned.

NetUserDel or Net32UserDel

NetUserDel or Net32UserDel

The NetUserDel API removes an account from the user account subsystem (UAS) database, ending all access to the resources in the system. Deleting an account also removes it from groups in the system.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

If the account being deleted is the last account in the database with administrator privilege, the error NERR_LastAdmin is returned and the delete function fails.

Directory and Security Server Only

Deletes a user from the DCE registry rather from the NET.ACC file. Authority is based on ACLs rather than privilege level. NERR_LastAdmin is never returned. If the last admin in a resource domain is deleted, RGYSYNC cannot delete the user from the NET.ACC file on the resource domain controller.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>    /* DSS only */

NetUserDel(pszTarget, pszUserID);                /* 16 bit */

Net32UserDel(pszTarget, pszUserID, pStatusbuf);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for the data types and descriptions of these parameters.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.

ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_UserLogon	2231	Deleting a user with a session is not allowed.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_LastAdmin	2452	The last administrator cannot be deleted.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead

- [DosWrite](#)

Related Information

For information about:

- Creating a user account in a UAS database, see [User - NetUserAdd](#).
- Listing user accounts on a server, see [User - NetUserEnum](#).
- Changing a user account, see [User - NetUserSetInfo](#).

NetUserEnum or Net32UserEnum

NetUserEnum or Net32UserEnum

The NetUserEnum API returns information about all accounts on a server. For DSS, the set of users returned depends on the scope of the cell.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. A user without Administrator authority can call this API only with level 0 or level 10. Administrator authority is required for full access.

Directory and Security Server Only

This API lists all DSS users within scope. It returns only the list of users that the caller has sufficient ACLs to read completely. See [Scoping Rules](#) for an explanation of how scoping works and about the special OTHUSERS entry.

Note: At level 0, the list might contain non-DSS users because the filter is based on name syntax rather than any DSS specific attributes.

Callers with **r** permission on a user can retrieve any available information level of detail. This is not restricted to ADMINS.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h> /* DSS only */

NetUserEnum(pszTarget, usLevel, buf, usBuflen,
            pusEntriesReturned, pusEntriesAvail); /* 16 bit */

Net32UserEnum(pszTarget, ulLevel, buf, ulBuflen,
              pulEntriesReturned, pulEntriesAvail, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, 2, 10 or 201, specifying which level of data structure to use, as described in [User Account Data Structures](#).

If *sLevel* is 1, 2, or 201, the password component of each data structure contains a string of spaces to maintain password security.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosRead

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total entries available. This technique is useful if you do not know the exact buffer size required.

For DSS, use the largest buffer size you can.

Related Information

For information about:

- Listing all groups to which a user belongs, see [User - NetUserGetGroups](#).
- Retrieving the status of a particular user account, see [User - NetUserGetInfo](#).

NetUserGetAppSel or Net32UserGetAppSel

NetUserGetAppSel or Net32UserGetAppSel

The NetUserGetAppSel API retrieves information about all specified types of applications contained in the user's desktop application folders.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

For NetUserGetAppSel to run successfully, one of the following must be true:

- The calling process has Administrator authority.
- The calling process has accounts operator privilege.
- The calling process has User privilege and is getting information about the logged-on user's desktop applications.

Domain control database subdirectories and files must exist for the user before NetUserGetAppSel is called. The NetUserDCDBInit API can be used to create the subdirectories and files.

Directory and Security Server Only

Any user with **r** permission on a user, retrieves the user's application selector list. Non-DSS instances of *app_sel_entry* Extended Registry Attributes (ERA) are ignored and not returned in the selector list. The format of the ERA allows for the possibility on non-DSS application selector entries.

Syntax

```
#include <netcons.h>
#include <dcdh.h>
#include <lsdceerr.h>    /* DSS only */

NetUserGetAppSel(pszTarget, pszUserID, usLevel, apptype,
                buf, usBuflen, pusBytesAvail);          /* 16 bit */

Net32UserGetAppSel(pszTarget, pszUserID, ulLevel, apptype,
                  buf, ulBuflen, pulBytesAvail,
                  pStatusbuf);                          /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in [User Application Data Structures](#).

apptype (16-bit unsigned short or 32-bit unsigned long) indicates the types of applications to retrieve, as defined in the DCDB.H header file:

SYMBOLIC CONSTANT	VALUE	MEANING
APP_DOS_PUBLIC	0x1	Public DOS applications
APP_OS2_PUBLIC	0x2	Public OS/2 applications
APP_OS2_PRIVATE	0x4	Private OS/2 application
APP_DOS_PRIVATE	0x8	Private DOS application

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.

NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr

Remarks

NetUserGetAppSel returns a buffer that points to an *app_sel_info_1* or *app_sel_info_200* structure. The structure element *asi1_count* or *asi200_count* indicates the number of *app_sel_list* or *app_sel_list_200* structures that immediately follow the *app_sel_info_1* or *app_sel_info_200* structure in the buffer. *asi1_count* or *asi200_count* also indicates the number of applications of the specified types that appear in the user's desktop application folder. This call is equivalent functionally to an Enum call.

If information is retrieved about both public and private applications, the return buffer contains information about all the private applications first, followed by public application information.

Related Information

For information about:

- Changing the set of applications that appear in a particular user's desktop application folders, see [User - NetUserSetAppSel](#).
- Retrieving information about a particular application definition, see [Application - NetAppGetInfo](#).
- Listing application definitions, see [Application - NetAppEnum](#).
- Creating an application definition, see [Application - NetAppAdd](#).
- Initializing the domain control database files for a particular user, see [User - NetUserDCDBInit](#).

NetUserGetGroups or Net32UserGetGroups

NetUserGetGroups or Net32UserGetGroups

The NetUserGetGroups API lists the names of all groups in the user accounts subsystem (UAS) database to which a particular user belongs. For DSS, this list depends on the scope of the cell. See [Scoping Rules](#) for details on scoping and the special OTHGRPS entry.

Users can use the NetUserGetGroups function on their user names.

The API returns the information in an array of *group_info_0* data structures. This call is equivalent functionally to an Enum call because it enumerates the groups of which a user is a member. For DSS, more than 254 groups may be returned.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. User authority is allowed limited access to this API. Administrator authority is required for full access.

Directory and Security Server Only

Any user can query the group membership list on any other user provided they have sufficient ACLs.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>      /* DSS only */

NetUserGetGroups(pszTarget, pszUserID, usLevel,
                buf, usBuflen,
                pulEntriesReturned, pulEntriesAvail);      /* 16 bit */

Net32UserGetGroups(pszTarget., pszUserID, ulLevel,
                  buf, ulBuflen,
                  pulEntriesReturned, pulEntriesAvail, pStatusbuf); /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszUserID (const unsigned char LSFAR *) points to the ASCIIZ string specifying the name of the user to search for in each group account.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 0, which specifies the data structure described in [Group Information Level 0](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosRead

Related Information

For information about retrieving the status of a user account, see [User - NetUserGetInfo](#).

NetUserGetInfo or Net32UserGetInfo

NetUserGetInfo or Net32UserGetInfo

The NetUserGetInfo API retrieves information about a particular account on a server. This includes information about the user's privileges, the home directory, and the status of an account.

To retrieve a user's comment or full name for a user ID, direct a NetUserGetInfo call to any server in the domain.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started.

A user without Administrator authority can call this API with levels 0, 10, and 11 only. Level 11 is available only for the account to which the user is logged on. Administrator authority is required for full access.

Directory and Security Server

Any user with **r** permission can call this API with levels 0, 1, 2, 10, 11, 201.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetUserGetInfo(pszTarget, pszUserID, usLevel,
               buf, usBuflen, pusBytesAvail); /* 16 bit */

Net32UserGetInfo(pszTarget, pszUserID, ulLevel,
                 buf, ulBuflen, pulBytesAvail,
                 pStatusbuf);               /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0, 1, 2, 10, or 11, specifying which level of data structure to use, as described in [User Account Data Structures](#).

When *sLevel* is 1 or 2, the password component of the *user_info* data structure contains a string of spaces to maintain password security.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been

		started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_BadUserDataInRgy.	7507	User data in registry is not valid for DSS
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Remarks

If you call this API with the buffer length parameter equal to zero, the API returns a value for total bytes available. This technique is useful if you do not know the exact buffer size required.

For DSS, use the largest size buffer you can.

Related Information

For information about:

- Changing a user's account, see [User - NetUserSetInfo](#).
- Retrieving a list of groups to which a user belongs, see [User - NetUserGetGroups](#).
- Retrieving information about all user accounts on a server, see [User - NetUserEnum](#).

SYMBOLIC CONSTANT	BIT MASK	MEANING
ALIAS TYPE FILE	0x0001	Network nickname used

ALIAS_TYPE_PRINTER	0x0002	Network nickname used
ALIAS_TYPE_SERIAL	0x0004	Network nickname used device

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_PATH_NOT_FOUND	3	The path was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- `DosLoadModule`
- `DosGetProcAddr`

Remarks

`NetUserLogonAsn` returns a buffer that points to a *logon_asn_info_1* or *logon_asn_info_200* structure. The structure element *lai1_count* or *lai200_count* indicates the number of *logon_asn_list* or *logon_asn_list_200* structures that immediately follow the *logon_asn_info_1* or *logon_asn_info_200* structure in the buffer. *lai1_count* or *lai200_count* also indicates the number of the user's logon assignments of specified types.

LAN Server and OS/2 Warp Server Only

You can assume a user's domain control database (DCDB) area has not been initialized if `NetUserGetLogonAsn` is called and the `ERROR_PATH_NOT_FOUND` code is returned.

This call is equivalent functionally to an Enum call.

Related Information

For information about:

- Initializing the domain control database for a particular user, see [User - NetUserDCDBInit](#).
- Changing a particular user's logon assignments, see [User - NetUserSetLogonAsn](#).
- Retrieving information about a particular alias definition, see [Alias - NetAliasGetInfo](#).
- Listing alias definitions, see [Alias - NetAliasEnum](#).
- Creating an alias definition, see [Alias - NetAliasAdd](#).

NetUserModalsGet or Net32UserModalsGet

NetUserModalsGet or Net32UserModalsGet

The `NetUserModalsGet` API gets global modals-related information for all users and groups.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. A user without Administrator authority can call this API at level 0 only. Administrator authority is required for full access. If a valid UAS database exists, you can call this API even when the UAS system is not active.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level. The user must have `r` permission on the registry policy and on the `ibm\ibm\lan\modals` organization.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>    /* DSS only */

NetUserModalsGet(pszTarget, usLevel, buf,
```

```

        usBuflen, pusBytesAvail);          /* 16 bit */
Net32UserModalsGet(pszTarget, ulLevel, buf,
        ulBuflen, pulBytesAvail, pStatusbuf); /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [User Modals Level 0](#) and [User Modals Level 1](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.

NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosRead

NetUserModalsSet or Net32UserModalsSet

NetUserModalsSet or Net32UserModalsSet

The NetUserModalsSet API sets global modals-related information for all users and groups.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation. Administrator authority is required to call this API. The Netlogon service cannot be running when this API is called at level 1 with *parmnum* set to PARMNUM_ALL or MODAL1_PARMNUM_ROLE.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>    /* DSS only */

NetUserModalsSet(pszTarget, usLevel, buf,
                usBuflen, parmnum);                /* 16 bit */

Net32UserModalsSet(pszTarget, ulLevel, buf,
                  ulBuflen, parmnum, pStatusbuf);  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) can be 0 or 1, specifying which data structure to use, as described in [User Modals Level 0](#) and [User Modals Level 1](#).

parmnum (16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a specific

field in the data structure is to be passed. If *parmnum* is 0, the entire data structure is sent, and *sLevel* can be either 0 or 1. Otherwise, *parmnum* specifies which field in the data structure is to be sent. All fields in the *user_modals_info_0* data structure, except *usrmod0_max_reserved1*, can be set with *parmnum*.

The following are the possible values for *parmnum*:

SYMBOLIC CONSTANT	VALUE	COMPONENT
PARMNUM_ALL	0	All elements
MODAL0_PARMNUM_MIN_LEN	1	usrmod0_min_passwd_len
MODAL0_PARMNUM_MAX_AGE	2	usrmod0_max_passwd_age
MODAL0_PARMNUM_MIN_AGE	3	usrmod0_min_passwd_age
MODAL0_PARMNUM_FORCEOFF	4	usrmod0_force_logoff
MODAL0_PARMNUM_HISTLEN	5	usrmod0_password_hist_len
MODAL0_PARMNUM_MAX_BADPW	6	usrmod0_maxbadpw
MODAL1_PARMNUM_ROLE	1	usrmod1_role
MODAL1_PARMNUM_PRIMARY	2	usrmod1_primary

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_WkstaNotStarted	2138	The Requester service has not been started.

NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_InvalidUASOp	2451	This operation is not permitted when the Netlogon service is running.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.

Other codes could be returned from the following functions:

- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosRead
- DosWrite

NetUserPasswordSet or Net32UserPasswordSet

NetUserPasswordSet or Net32UserPasswordSet

The NetUserPasswordSet API changes a user's password.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. This API does not have any access authority requirements.

Syntax

```

#include <netcons.h>
#include <access.h>
#include <lsdeerr.h>      /* DSS only */

NetUserPasswordSet(pszTarget, pszUserID,
                  pszOldPassword, pszNewPassword);      /* 16 bit */

Net32UserPasswordSet(pszTarget, pszUserID,
                   pszOldPassword, pszNewPassword, pStatusbuf); /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszOldPassword (unsigned char LSFAR *) points to an ASCIIZ string specifying the user's current password.

pszNewPassword (unsigned char LSFAR *) points to an ASCIIZ string specifying a new password for the user.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_PASSWORD	86	The specified password is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadPassword	2203	The specified password is not valid.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.

NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_AccountExpired	2239	The user account has expired.
NERR_PasswordCantChange	2243	This password cannot change.
NERR_PasswordHistConflict	2244	This password cannot be used now.
NERR_PasswordTooShort	2245	The password is shorter than required.
NERR_PasswordTooRecent	2246	The password is too recent to change.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosDevIOCtl
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Remarks

Passwords are case-sensitive.

The password is set to *pszNewPassword* only if all of the following are true:

- *pszOldPassword* matches the current password for the user.
- *pszNewPassword* is not the same as the current password.
- *pszNewPassword* is not the same as any of the historic passwords tracked by *usrmod0_password_hist_len* field in the *user_modals_info_0* data structure.

This call allows users to change their own passwords without administrator privilege. Password update restrictions apply.

A user with administrator privilege who does not know the old password still can change a user's password by using *NetUserSetInfo* instead of this API.

With a call to the domain controller of a single system image (SSI) domain to which the user currently is logged on, NetUserPasswordSet also sets the current default password of the requester. This allows the user to connect to all servers in the domain, because the new password should replicate rapidly throughout the domain.

Special Server Actions

When used remotely, a call to the NetUserPasswordSet API still succeeds when the user's current password has expired. If the password for an account has expired but the account otherwise could log on, the server permits a NetUserPasswordSet call to succeed.

Related Information

For information about setting a password, see [User - NetUserSetInfo](#).

NetUserSetAppSel or Net32UserSetAppSel

NetUserSetAppSel or Net32UserSetAppSel

The NetUserSetAppSel API sets the list of applications contained in the specified user's desktop application folders.

To add a single application to an existing application selector, an application can:

1. Call NetUserGetAppSel, which returns an *app_sel_info_1* header structure followed by zero or more *app_sel_list* structures, one for each application available to the specified *pszUserID*.
2. Add another *app_sel_list* structure to append the new application to the returned structure.
3. Be sure to increase the *asi1_count* field in the data structure by the number of new entries; in this case, by 1.
4. Note that the buffer must be large enough to hold another *app_sel_list* structure. If it is not, you must increase the *usBuflen* parameter accordingly.
5. Pass the new data structure to the NetUserSetAppSel API.

To delete all applications from the user's selectors, call the NetUserSetAppSel API with *sLevel* parameter set to 1 and the data structure *asi1_count* field set to 0.

Restrictions

LAN Server and OS/2 Warp Server Only

DCDB subdirectories and files must exist for the user before NetUserSetAppSel is called. To create these files initially, use the API described [User - NetUserDCDBInit](#).

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

For NetUserSetAppSel to run successfully, one of the following must be true:

- The calling process has Administrator authority.
- The calling process has accounts operator privilege.
- The calling process has User authority and is setting information for the logged-on user.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level. DSS also supports level 200, which allows setting all of the user's application selector entries regardless of what resource domain they are in.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h> /* DSS only */

NetUserSetAppSel(pszTarget, pszUserID, usLevel,
```

```

        buf, usBuflen);                                /* 16 bit */

Net32UserSetAppSel(pszTarget, pszUserID, ulLevel,
        buf, ulBuflen, pStatusbuf); /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszServername (const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the domain controller for the domain in which *pszUserID* is defined. It can be NULL only if the calling application is running on the domain controller.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1, which specifies the data structure described in [User Application Data Structures](#).

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_DATA	13	The specified data is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_AppNotFound	2793	The application was not found.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.

NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_AppNotInResDom	7521	The application is not in the specified resource domain.

Other codes could be returned from the following functions:

- DosLoadModule
- DosGetProcAddr
- DosOpen2
- DosWrite

Related Information

For information about:

- Retrieving information about the applications that appear in a particular user's desktop application folders, see [Application - NetAppGetInfo](#).
- Listing application definitions, see [Application - NetAppEnum](#).

NetUserSetGroups or Net32UserSetGroups

NetUserSetGroups or Net32UserSetGroups

The NetUserSetGroups API sets the groups of which a user is a member. In DSS users can belong to more than 254 groups.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started. Administrator authority is required to call this API.

Directory and Security Server Only.

Authority is based on ACLs rather than privilege level.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetUserSetGroups(pszTarget pszUserID, usLevel,
                 buf, usBuflen, numentries);    /* 16 bit */

Net32UserSetGroups(pszTarget, pszUserID, ulLevel,
                  buf, ulBuflen, numentries,
                  pStatusbuf);                  /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszUser/D</i>	(const unsigned char LSFAR *) points to the ASCIIZ string specifying the name of the user whose group is being modified.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 0, specifying the data structure described in Group Information Level 0 .
<i>numentries</i>	(16-bit unsigned short or 32-bit unsigned long) specifies the number of data structures to be passed with this API call.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_INVALID_NAME	123	There is an incorrect character or incorrectly formed file system name.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_CanNotGrowSegment	2233	It is not possible to enlarge the UAS cache segment.

NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

NetUserSetInfo or Net32UserSetInfo

NetUserSetInfo or Net32UserSetInfo

The NetUserSetInfo API modifies user information, such as passwords and access authority.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called from DLS and OS/2 workstations, but only DLS workstations can issue this call to a remote LAN Server workstation. DLS workstations can issue this call locally only if the Peer service is started.

Users with administrator authority have full to access the NetUserSetInfo APIs and can set fields for any user account, such as required passwords or access privileges. Most of the functions provided by NetUserSetInfo require Administrator authority; for instance, changing a password for any account, including one's own, requires administrator authority.

However, NetUserSetInfo does allow User authority access to three fields in the UAS database, but only for the account of the user issuing the call. These fields are set one at a time according to the *parmnum* parameter, as described below.

NULL_USERSETINFO_PASSWD must be used with the *parmnum* 0 option if the user does not want the password to be changed.

This API cannot change the privilege of or disable the last account in the database with administrator privilege; if you attempt either of these, the function fails and the error NERR_LastAdmin is returned.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level. Permissions vary according to *parmnum*. This API never returns NERR_LastAdmin.

NULL_USERSETINFO_PASSWD must be used with the *parmnum* 0 option if the user does not want the password to be changed.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>          /* DSS only */

NetUserSetInfo(pszTarget, pszUserID, usLevel,
               buf, usBuflen, parmnum);    /* 16 bit */

Net32UserSetInfo(pszTarget, pszUserID, ulLevel,
                 buf, ulBuflen, parmnum,
                 pStatusbuf);             /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

- sLevel* or *ulLevel*
- (16-bit short or 32-bit unsigned long) can be 1 or 2 or for DSS level 201 specifying which data structure to use, as described in [User Account Information Level 1](#) and [User Account Information Level 2](#).
- parmnum*
- (16-bit short or 32-bit unsigned long) specifies whether the entire data structure or only a single field in the data structure is to be passed. If this parameter is 0, the entire data structure is passed. Otherwise, *parmnum* can be set to one of the following values to pass only a single field.

SYMBOLIC CONSTANT	VALUE	DATA STRUCTURE FIELD
PARMNUM_ALL	0	user_info_X
PARMNUM_NAME	1	usriX_name
PARMNUM_PAD	2	usriX_pad_1
PARMNUM_PASSWD	3	usriX_password
PARMNUM_PRIV	5	usriX_priv
PARMNUM_DIR	6	usriX_home_dir
PARMNUM_COMMENT	7	usriX_comment
PARMNUM_USER_FLAGS	8	usriX_flags
PARMNUM_SCRIPT_PATH	9	usriX_script_path
PARMNUM_AUTH_FLAGS	10	usriX_auth_flags
PARMNUM_FULL_NAME	11	usriX_full_name
PARMNUM_USR_COMMENT	12	usriX_usr_comment
PARMNUM_PARMS	13	usriX_parms

PARMNUM_WORKSTATIONS	14	usriX_workstations
PARMNUM_ACCT_EXPIRES	17	usriX_acct_expires
PARMNUM_MAX_STORAGE	18	usriX_max_storage
PARMNUM_UNITS_PER_WEEK	19	usriX_units_per_week
PARMNUM_LOGON_HOURS	20	usriX_logon_hours
PARMNUM_LOGON_SERVER	23	usriX_logon_server
PARMNUM_COUNTRY_CODE	24	usriX_country_code
PARMNUM_CODE_PAGE	25	usriX_code_page
PARMNUM_PASSWD_EXPIRED	100	usriX_password_age
PARMNUM_DEFAULT_REALM	200	usriX_default_realm
PARMNUM_PRIMARY_GROUP	201	usriX_dce_primary_group
PARMNUM_POLICY_GROUP	202	usriX_dce_policy_group
SYMBOLIC CONSTANT	VALUE	DATA STRUCTURE FIELD
PARMNUM_PWSTRENGTH_SERVER	204	usriX_pwstrength_server

NOTES:

X = 1 or 2

When parmnum is set to PARMNUM_PASSWD_EXPIRED, the password changed in one of two ways:

1. If 0 is passed in the buffer, the password age is revived.
2. If a non-zero value is passed in the buffer, the password expires and the user must change the password at next logon.

The password age must be expired before it can be revived. To revive an active password, NetUserSetInfo has no effect.

Administrator authority is required for all parmnum values. The following fields are set for a user's own account:

usriX_usr_comment	(parmnum = 7)
usriX_parms	(parmnum = 13)
usriX_country_code	(parmnum = 24)
usriX_code_page	(parmnum = 25)

The following symbolic constants support DLS Peer services:

```

PARMNUM_ALL
PARMNUM_PASSWD
PARMNUM_PRIV
PARMNUM_COMMENT
PARMNUM_USER_FLAGS
PARMNUM_AUTH_FLAGS
PARMNUM_FULL_NAME
PARMNUM_USR_COMMENT
PARMNUM_ACCT_EXPIRES

```

PARMNUM_COUNTRY_CODE
PARMNUM_CODE_PAGE

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_NOT_SUPPORTED	50	This request is not supported by the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.
ERROR_BAD_NET_NAME	67	This network name cannot be found.
ERROR_INVALID_PARAMETER	87	At least one parameter value is not valid.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_RemoteOnly	2106	This operation is not supported on workstations.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transactions.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadPassword	2203	The specified password is not valid.
NERR_ACFNotFound	2219	The NET.ACC file is missing.
NERR_GroupNotFound	2220	The group does not exist.
NERR_UserNotFound	2221	The user name cannot be found.

NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_PasswordTooShort	2245	The password is shorter than required.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_LastAdmin	2452	The last administrator cannot be deleted.
NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosDevIOCtl
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

Related Information

For information about a particular user name on a server, see [User - NetUserGetInfo](#).

NetUserSetLogonAsn or Net32UserSetLogonAsn

NetUserSetLogonAsn or Net32UserSetLogonAsn

The NetUserSetLogonAsn API sets logon assignments for a user.

To add a single logon assignment to an existing user logon, the application must:

1. Call NetUserGetLogonAsn.
2. Append a *logon_asn_list* data structure (with the new logon assignment) to the buffer returned from NetUserGetLogonAsn. (The buffer must be large enough to hold another *logon_asn_list*. If it is not large enough, increase the value of *usBuflen* and reallocate the buffer.)
3. Increase the *lail_count* field by 1.
4. Call NetUserSetLogonAsn.

To delete all of a user's logon assignments, *buf* must point to a *logon_asn_info_1* structure with its *lail_count* element set to 0.

Restrictions

LAN Server and OS/2 Warp Server Only

Domain control database subdirectories and files must exist for the user before NetUserSetLogonAsn is called. You can use the NetUserDCDBInit API to create the subdirectories and files.

This API can be called from DLS and OS/2 workstations, but DLS workstations can issue this call only to a remote LAN Server workstation.

For NetUserSetLogonAsn to run successfully, one of the following must be true:

- The calling process has Administrator authority.
- The calling process has accounts operator privilege.
- The calling process has User authority and is setting information about the logged-on user's logon assignments.

The *la_type* field of the *logon_asn_list* data structure is ignored for this API.

Directory and Security Server Only

Authority is based on ACLs rather than privilege level.

DSS also supports level 200 which allows setting logon assignments for multiple resource domains at the same time.

Syntax

```
#include <netcons.h>
#include <dcdb.h>
#include <lsdceerr.h>    /* DSS only */

NetUserSetLogonAsn(pszTarget, pszUserID, usLevel,
                  buf, usBuflen);    /* 16 bit */

Net32UserSetLogonAsn(pszTarget, pszUserID, ulLevel,
                    buf, ulBuflen, pStatusbuf);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

<i>pszServername</i>	(const unsigned char LSFAR *) points to an ASCIIZ string containing the name of the domain controller of a domain in which <i>pszUserID</i> is defined. It can be null only if the calling application is running on a domain controller.
<i>sLevel</i> or <i>ulLevel</i>	(16-bit short or 32-bit unsigned long) must be 1 or 200, which specifies the data structure described in Logon Assignment Data Structure .

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_FILE_NOT_FOUND	2	The file was not found.
ERROR_TOO_MANY_OPEN_FILES	4	The maximum number of open files was exceeded.
ERROR_ACCESS_DENIED	5	Administrator privilege is required.
ERROR_NOT_ENOUGH_MEMORY	8	Sufficient memory is not available.
ERROR_INVALID_DATA	13	The specified data is not valid.
ERROR_ALREADY_ASSIGNED	85	Duplicate redirection.
ERROR_INVALID_LEVEL	124	The ul/usLevel parameter is not valid.
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_AliasNotFound	2783	The alias does not exist.
NERR_DCDBError	2786	A domain control database file is unreadable or cannot be accessed at this time.
NERR_NotPrimaryDCDB	2795	An attempt was made to access a domain control database file on a machine that is not the domain controller.
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_AliasNotInResDom	7522	The alias is not defined to the specified resource domain.

Other codes could be returned from the following functions:

- [DosLoadModule](#)
- [DosGetProcAddr](#)

Related Information

For information about:

- Initializing the domain control database for a particular user, see [User - NetUserDCDBInit](#).
- Retrieving a particular user's logon assignments, see [User - NetUserGetLogonAsn](#).
- Changing information for a particular alias definition, see [Alias - NetAliasSetInfo](#).

NetUserValidate2 or Net32UserValidate2

NetUserValidate2 or Net32UserValidate2

The NetUserValidate2 API validates a user ID with its password, and checks whether there are any logon restrictions for this user account. If the specified user account is not found, this API attempts to validate the guest account, instead. If this happens, all verification checks will be based only on the guest account. If the call completes successfully, a *user_logon_info_1* structure is returned in the same input buffer and the code is set appropriately.

The NetUserValidate2 API replaces the NetUserValidate API, which is now obsolete.

Restrictions

LAN Server and OS/2 Warp Server Only

This API can be called only from an OS/2 workstation.

Syntax

```
#include <netcons.h>
#include <access.h>
#include <lsdceerr.h>    /* DSS only */

NetUserValidate2(pszTarget, sLevel, buf,
                usBuflen, reserved2, pusBytesAvail);    /* 16 bit */

Net32UserValidate2(pszTarget, ulLevel, buf,
                  ulBuflen, reserved2, pulBytesAvail,
                  pStatusbuf);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

sLevel or *ulLevel* (16-bit short or 32-bit unsigned long) must be 1.

Note: When an application issues the NetUserValidate2 call, it passes a *user_logon_req_1* data structure, described in [User Logon Request Level 1](#). The returned data structure, however, is *user_logon_info_1*, described in [User Logon Level 1](#).

reserved2 (16-bit unsigned short or 32-bit unsigned long) must be 0.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_INVALID_PASSWORD	86	The specified password is not valid.
ERROR_INVALID_LEVEL	124	The sLevel parameter is not valid.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_InvalidDatabase	2247	The UAS database file is damaged.

Other codes could be returned from the following functions:

- DosAllocSeg
- DosChgFilePtr
- DosFsCtl
- DosGetShrSeg
- DosNewSize
- DosQFileInfo
- DosRead
- DosWrite

User Profile Management Category

This category includes the following APIs:

UPM - UPMELOCL
UPM - UPMELOCU
UPM - UPMEULGF
UPM - UPMEULGN
UPM - UPMEUSRL

DOS Considerations

DOS does not support the APIs in this category.

UPMELOCL or U32ELOCL

UPMELOCL or U32ELOCL

The UPMELOCL API invokes User Profile Management to display a local logon window. This allows users to enter their user IDs and passwords to perform a local logon to the system.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

If a user already is logged on locally, the return code UPM_OK is returned. If a user already is logged on to the domain, this function attempts to use the user ID and password to log on locally. If that attempt is unsuccessful, the function displays the local logon window with the user ID in the **USER ID** field. The password is not displayed.

Syntax

```
#include <upm.h>

LSINT LSFAR LSPAS
UPMELOCL(pszUserID, pAuthorityLevel);    /* 16 bit */

LSINT LSFAR LSPAS
U32ELOCL(pszUserID, pAuthorityLevel);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pAuthorityLevel (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to the level of access authority for this user, which can be UPM_USER, UPM_LOCAL_ADMIN, or UPM_ADMIN.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	MEANING
UPM_OK	The user was logged on successfully.
UPM_LOG_CANCEL	The user cancelled from the logon panel.
UPM_LOG_INPROC	Another logon was being processed; the logon attempt was not successful.
UPM_SYS_ERROR	A system error occurred.

UPMELOCU or U32ELOCU

UPMELOCU or U32ELOCU

The UPMELOCU API retrieves an ID of a local user that already is logged on to the system.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

If the LOGON /O=MULTI option has been issued, the return code is UPM_NOT_LOGGED.

Syntax

```
#include <upm.h>

LSINT LSFAR LSPAS
UPMELOCU(pszUserID, pLocalAuthority);    /* 16 bit */

LSINT LSFAR LSPAS
U32ELOCU(pszUserID, pLocalAuthority);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pLocalAuthority (16-bit unsigned short LSFAR * or 32-bit unsigned long LSFAR *) points to this user's level of local access authority for an HPFS386 Server with local security installed. This can be UPM_USER, UPM_LOCAL_ADMIN, or UPM_ADMIN.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	MEANING
UPM_OK	The user was logged on successfully.
UPM_NOT_LOGGED	The user has not logged on locally.
UPM_LOG_INPROC	A logon is in process.
UPM_SYS_ERROR	A system error occurred.

UPMEULGF or U32EULGF

UPMEULGF or U32EULGF

The UPMEULGF API logs a user off the system.

To use this API, you must be linked to the UPM.LIB library. In addition, you should have at least 8KB of free stack space.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

All parameters must be valid; if not, an error is returned.

For the *remotetype* of UPM_DOMAIN and UPM_DOMAIN_MAX_FORCE, the *pszRemotename* is ignored if the logon was performed with no verification or local verification.

Syntax

```
#include <upm.h>
```

```

LSINT LSFAR LSPAS
UPMEULGF(pszUserID, pszRemotename,
         remotetype);    /* 16 bit */

LSINT LSFAR LSPAS
U32EULGF(pszUserID, pszRemotename,
         remotetype);    /* 32 bit */

```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszRemotename (unsigned char LSFAR *) points to the name of the node or domain for which the user is logging off.

remotetype (16-bit unsigned short or 32-bit unsigned long) is the type of node for *pszRemotename*. The following are the possible values:

Symbolic Constant	VALUE	MEANING
UPM_LOCAL	1	The local node.
UPM_DNODE	2	Specifies that remotename is node from which the user was logged off.
UPM_DOMAIN	3	Specifies that remotename is domain. If the domain user logged off also has 386 HPFS access, user logged on is assigned local security access.
UPM_DOMAIN_MAX_FORCE	4	Same meaning as UPM_DOMAIN, all connections are forced to be local.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	MEANING
UPM_OK	The user was logged off successfully.
UPM_BAD_TYPE	The remote type specified was not valid.
UPM_NOT_LOGGED	The specified user ID was not logged on.
UPM_OPEN_SESSIONS	The domain logoff failed because a domain logon has active open sessions.
UPM_SYS_ERROR	An unexpected system error occurred.
UPM_BAD_PARAMETER	The user ID or remote name was not valid.
UPM_ACTIVE	The domain logon or logoff failed; a domain logon, logoff, or the LAN Requester graphical user interface is active.

UPMEULGN or U32EULGN

UPMEULGN or U32EULGN

The UPMEULGN API logs on a user. This function supports a logon prompt and registers user IDs like the command-line logon.

To use the UPMEULGN API, you must be linked to the UPM.LIB library. In addition, you need at least 8KB of free stack space.

Restrictions

A call to this API can be issued only from an OS/2 application to the local computer.

Syntax

```
#include <upm.h>

LSINT LSFAR LSPAS
UPMEULGN(pszUserID, pszPassword, pszRemotename,
         remotetype, flags);    /* 16 bit */

LSINT LSFAR LSPAS
U32EULGN(pszUserID, pszPassword, pszRemotename,
         remotetype, flags);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszPassword (unsigned char LSFAR *) points to an ASCIIZ string containing the password of the user name, obtained by an application's request to the user. A null pointer or string indicates that no password is needed. After a user is logged on to a requester, the *pszPassword* and *pszUserID* are verified again by LAN Server whenever the same requester attempts to access a remote resource.

pszRemotename (unsigned char LSFAR *) points to the name identifying the node or domain for which the user ID and password are required. If *remotetype* is UPM_LOCAL, *pszRemotename* is ignored.

remotetype (16-bit unsigned short or 32-bit unsigned long) is the type of node named in *pszRemotename* for which the user ID and password are required. The following are the possible values:

SYMBOLIC CONSTANT	VALUE	MEANING
UPM_LOCAL	1	Indicates the local node. If installed, a user logging on to an assigned 386-HPFS local access domain logon has been performed.
UPM_DNODE	2	Specifies that remotename is an LU 6.2 node to which the user is logging on.
UPM_DOMAIN	3	Specifies that remotename is a domain. If the 386 HPFS is a domain user being logged on to, 386-HPFS access. flags only for remotetype of UPM_DOMAIN.

Bits 2-15 are reserved and must be 0.

flags

(16-bit unsigned short or 32-bit unsigned long) is a bit mask for LAN Server logon verification. Valid values for bits 0 and 1 are:

SYMBOLIC CONSTANT	BIT MASK	MEANING
UPM_FL_LOCVER	0x01	Verification of logon at the station.
UPM_FL_NOVER	0x02	No logon verification required.
UPM_FL_DOMVER	0x03	Verification of logon to the domain.

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	MEANING
UPM_OK	The user was logged on successfully.
UPM_LOG_INPROC	Another logon was being processed; the logon attempt was not successful.
UPM_BAD_TYPE	The remote type specified was not valid.
UPM_NOMEM	Insufficient memory is available to process the request.
UPM_LOG_FILE_NOT_FOUND	A required logon program was not found.
UPM_FAIL_SECURITY	The user ID or password could not be validated.
UPM_ULP_LOADED	The local logon failed; a local logon with a user logon profile is already active.
UPM_PASSWORD_EXP	The logon failed; the user's password is expired.
UPM_UNAVAIL	The logon failed; the remote node or domain could not be contacted to process the logon request.
UPM_ACTIVE	The domain logon or logoff failed; a domain logon, logoff, or the LAN Requester graphical user interface is active.
UPM_SS_PWDEXPWARNING	The local logon succeeded. The user's password is expired.
UPM_SS_BUSY	The local logon failed; the secure shell was busy.
UPM_SS_DEAD	The local logon failed; the secure shell has stopped.
UPM_LOGGED	The domain logon failed because a domain logon has already occurred.
UPM_SYS_ERROR	An unexpected system error occurred.
UPM_BAD_PARAMETER	The user ID, password, or remote name was not valid.

Remarks

When *remotetype* is UPM_LOCAL or UPM_DOMAIN, the user ID and password are verified immediately against the existing users. When *remotetype* is UPM_DNODE, the user ID and password are verified on the attempt to attach to the remote node.

UPM_FL_LOCVER specifies local verification for the user ID and password; UPM_FL_NOVER specifies no verification; and UPM_FL_DOMVER specifies domain verification. The *remotename* must be NULL if either UPM_FL_LOCVER or UPM_FL_NOVER is specified.

If *flags* is 0, the verification for the UPM_DOMAIN *remotetype* logon is the type specified by the wrkhueristic parameter number 3 in the IBMLAN.INI file.

UPMEUSRL or U32EUSRL

UPMEUSRL or U32EUSRL

The UPMEUSRL API returns a list of logged-on user IDs. You can use this API to get all user IDs logged on at a particular local machine, without having to specify a server name. The type of user IDs returned depends on the *remotetype* input parameter.

Restrictions

- A call to this API can be issued only from an OS/2 application to the local computer.
- To use this API, you must be linked to the UPM.LIB library.

Syntax

```
#include <upm.h>

LSINT LSFAR LSPAS
UPMEUSRL(pszRemotename, remotetype, buf,
        usBuflen, pusEntriesReturned, pusEntriesAvail);    /* 16 bit */

LSINT LSFAR LSPAS
U32EUSRL(pszRemotename, remotetype, buf,
        ulBuflen, pulEntriesReturned, pulEntriesAvail);    /* 32 bit */
```

Parameters

See [Common Parameter Definitions](#) for descriptions of parameters not defined here.

pszRemotename (unsigned char LSFAR *) points to the name identifying the node or domain from which the logged-on user IDs are requested. If *remotetype* is UPM_LOCAL, UPM_LOCAL_HPFS, UPM_DOMAIN, or NULL, *pszRemotename* is ignored.

remotetype (16-bit short or 32-bit unsigned long) specifies about which users to return information. The following are the possible values:

SYMBOLIC CONSTANT	VALUE	MEANING
UPM_LOCAL	1	User IDs logged on to the local n

UPM_DNODE	2	User IDs logged on to remote node
UPM_DOMAIN	3	All LAN Server Logons, whether local or remote, that have been invalidated.
UPM_LOCAL_HPFS	21	User IDs logged on to the high performance file system
UPM_ALL	22	All user IDs

buf (unsigned char LSFAR *) points to a UPM logon data structure with the following format.

UPM Logon Data Structure

```
struct upm_user_logon {
    unsigned char    userid [UPM_UIDLEN + 1];
    short            remotetype;
    unsigned char    remotename [UPM_REMLEN + 1];
    LSINT            sessionid;
};
```

Return Codes

The following table lists the return codes most significant to this API. (See [API Return Codes](#) for a complete list of return codes.)

SYMBOLIC CONSTANT	MEANING
UPM_OK	The call was completed successfully.
UPM_ERROR_MORE_DATA	All the available UPM_USER_LOGON entries did not fit into buf.
UPM_BAD_TYPE	The remote type specified is not valid.
UPM_BAD_PARAMETER	The remote name is not valid.
UPM_NOMEM	Insufficient memory is available to process the request.
UPM_SYS_ERROR	An unexpected system error occurred.

API Return Codes

This appendix lists the value of each LAN Server API return code in numeric order, as well as the *symbolic constant*, or phrase, assigned to that code. This information is defined in the \BMLAN\NETSRC\H\NETERR.H header file.

Successful Completion of an API Call

A LAN Server API function that encounters no error returns this to the application that issued the call.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_Success	0	No errors were encountered.

Redirector

A LAN Server API function returning error codes 50-88 or 230-240 encountered an error with the redirector either directly or when using a remote API call.

SYMBOLIC CONSTANT	VALUE	MEANING
ERROR_NOT_SUPPORTED	50	The network request is not supported.
ERROR_REM_NOT_LIST	51	This remote computer is not listening.
ERROR_DUP_NAME	52	A duplicate name exists on the network.
ERROR_BAD_NETPATH	53	The network path cannot be found.
ERROR_NETWORK_BUSY	54	The network is busy.
ERROR_DEV_NOT_EXIST	55	This device does not exist on the network.
ERROR_TOO_MANY_CMDS	56	The IBM NetBIOS command limit has been exceeded.
ERROR_ADAP_HDW_ERR	57	A network adapter hardware error has occurred.
ERROR_BAD_NET_RESP	58	The network has responded incorrectly.
ERROR_UNEXP_NET_ERR	59	An unexpected network error has occurred.
ERROR_BAD_REM_ADAP	60	The remote adapter being used is incompatible.
ERROR_PRINTQ_FULL	61	The printer queue is full.
ERROR_NO_SPOOL_SPACE	62	There is not enough memory available for the requested print file.
ERROR_PRINT_CANCELED	63	The requested print file has been canceled.
ERROR_NETNAME_DELETED	64	The network name was deleted.
ERROR_NETWORK_ACCESS_DENIED	65	Network access is denied.

ERROR_BAD_DEV_TYPE	66	This network device type is incorrect.
ERROR_BAD_NET_NAME	67	This network name cannot be found.
ERROR_TOO_MANY_NAMES	68	The network name limit has been exceeded.
ERROR_TOO_MANY_SESS	69	The IBM NetBIOS session limit has been exceeded.
ERROR_SHARING_PAUSED	70	File sharing has been temporarily paused.
ERROR_REQ_NOT_ACCEP	71	This request is not accepted by the network.
ERROR_REDIR_PAUSED	72	Print or disk redirection is temporarily paused.
ERROR_NET_WRITE_FAULT	88	A network fault occurred.
ERROR_BAD_PIPE	230	This is a non-existent pipe or an operation that is not valid
ERROR_PIPE_BUSY	231	The specified pipe is busy.
ERROR_NO_DATA	232	There is no data on a non-blocking read.
ERROR_PIPE_NOT_CONNECTED	233	The pipe was disconnected by the server.
ERROR_MORE_DATA	234	Additional data is available, but the buffer is too small.
ERROR_VC_DISCONNECTED	240	The session was canceled.

Network Utilities

A LAN Server API function returning error codes 2102-2143 encountered an error with a network utility.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NetNotStarted	2102	The redirector NETWKSTA.200 has not been started.
NERR_UnknownServer	2103	The server cannot be located.
NERR_ShareMem	2104	An internal error occurred-the network cannot access a shared memory segment.
NERR_NoNetworkResource	2105	A network resource shortage occurred.
NERR_RemoteOnly	2106	This operation is not supported on workstations.

NERR_DevNotRedirected	2107	The device is not connected.
NERR_ServerNotStarted	2114	The Server service has not been started.
NERR_ItemNotFound	2115	The device queue is empty.
NERR_UnknownDevDir	2116	The device or directory does not exist.
NERR_RedirectedPath	2117	The operation is not valid on a redirected device.
NERR_DuplicateShare	2118	The name has already been shared.
NERR_NoRoom	2119	The server is currently out of the requested resource.
NERR_TooManyItems	2121	The requested addition of an item exceeds the maximum.
NERR_BufTooSmall	2123	The buffer is too small for fixed-length data.
NERR_RemoteErr	2127	A remote API error has occurred.
NERR_LanIniError	2131	An error occurred when opening or reading the IBMLAN.INI file.
NERR_OS2IoctlError	2134	An internal error occurred when calling the workstation driver.
NERR_NetworkError	2136	A general network error has occurred.
NERR_WkstaNotStarted	2138	The Requester service has not been started.
NERR_BrowserNotStarted	2139	The requested information is not available.
NERR_InternalError	2140	An internal error has occurred.
NERR_BadTransactConfig	2141	The server is not configured for transaction.
NERR_InvalidAPI	2142	The requested API is not supported on the remote server.
NERR_BadEventName	2143	The event name is not valid.

Configuration

A LAN Server API function returning error codes 2146-2149 encountered an error with configuration.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_CfgCompNotFound	2146	The program could not find the specified component in the IBMLAN.INI file.

NERR_CfgParamNotFound	2147	The program could not find the specified parameter in the IBMLAN.INI file.
NERR_LineTooLong	2149	A line in the IBMLAN.INI file is too long.

Spooler

A LAN Server API function returning error codes 2150-2168 encountered an error while changing a network spooler.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_QNotFound	2150	The printer queue does not exist.
NERR_JobNotFound	2151	The print job does not exist.
NERR_DestNotFound	2152	The print destination cannot be found.
NERR_DestExists	2153	The printer destination already exists.
NERR_QExists	2154	The printer queue already exists.
NERR_QNoRoom	2155	No more printer queues can be added.
NERR_JobNoRoom	2156	No more print jobs can be added.
NERR_DestNoRoom	2157	No more print destinations can be added.
NERR_DestIdle	2158	This print destination is idle and cannot accept control operations.
NERR_DestInvalidOp	2159	This print destination request contains a control function that is not valid.
NERR_ProcNoRespond	2160	The printer processor is not responding.
NERR_SpoolerNotLoaded	2161	The spooler service has not been started.
NERR_QInvalidState	2163	This operation cannot be performed on the printer queue in its current state.
NERR_JobInvalidState	2164	This operation cannot be performed on the print job in its current state.
NERR_SpoolNoMemory	2165	A spooler memory allocation failure has occurred.
NERR_DriverNotFound	2166	The device driver does not exist.
NERR_DataTypeInvalid	2167	The data type is not supported by the processor.
NERR_ProcNotFound	2168	The print processor is not installed.

Service

A LAN Server API function returning error codes 2180-2191 encountered an error with one of the network services.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_ServiceTableLocked	2180	The service does not respond to control actions.
NERR_ServiceTableFull	2181	The service table is full.
NERR_ServiceInstalled	2182	The requested service has already been started.
NERR_ServiceEntryLocked	2183	The service does not respond to control actions.
NERR_ServiceNotInstalled	2184	The service has not been started.
NERR_BadServiceName	2185	The service name is not valid.
NERR_ServiceCtlTimeout	2186	The service is not responding to the control function.
NERR_ServiceCtlBusy	2187	The service control is busy.
NERR_BadServiceProgName	2188	The IBMLAN.INI file contains a service program name that is not valid.
NERR_ServiceNotCtrl	2189	The service cannot be controlled in its present state.
NERR_ServiceKillProc	2190	The service was ended abnormally.
NERR_ServiceCtlNotValid	2191	The requested pause or stop is not valid for this service.

Requester

A LAN Server API function returning error codes 2200-2217 encountered an error with a requester.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_AlreadyLoggedIn	2200	This workstation is already logged on.
NERR_NotLoggedIn	2201	This workstation has not been logged on yet.
NERR_BadUsername	2202	The pszUserID or pszGroupID parameter is not valid.
NERR_BadPassword	2203	The specified password is not valid.

NERR_UnableToAddName_W	2204	The logon processor did not add the message alias.
NERR_UnableToAddName_F	2205	The logon processor did not add the message alias.
NERR_UnableToDelName_W	2206	The logoff processor did not delete the message alias.
NERR_UnableToDelName_F	2207	The logoff processor did not delete the message alias.
NERR_LogonsPaused	2209	Network logons are paused.
NERR_LogonServerConflict	2210	A centralized logon server conflict has occurred.
NERR_LogonNoUserPath	2211	The server is configured without a valid user path.
NERR_LogonScriptError	2212	An error occurred while loading or running the logon script.
NERR_StandaloneLogon	2214	The logon server was not specified-standalone logon occurs.
NERR_LogonServerNotFound	2215	The logon server cannot be found.
NERR_LogonDomainExists	2216	There is already a logon domain for this computer.
NERR_NonValidatedLogon	2217	The logon server could not validate the logon.

Access, User, and Group

A LAN Server API function returning error codes 2219-2249 encountered an error while requesting or modifying information concerning network privileges.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_AccountUndeletable	2218	The user account is marked undeletable.
NERR_ACFNotFound	2219	The accounts file NET.ACC cannot be found.
NERR_GroupNotFound	2220	The group does not exist.
NERR_UserNotFound	2221	The user name cannot be found.
NERR_ResourceNotFound	2222	The netname cannot be found.
NERR_GroupExists	2223	The group name is already in use.
NERR_UserExists	2224	The user account already exists.

NERR_ResourceExists	2225	The resource permission list already exists.
NERR_NotPrimary	2226	The UAS database is replicant and does not allow updates.
NERR_ACFNotLoaded	2227	The UAS database has not been started.
NERR_ACFNoRoom	2228	There are too many names in the access control file.
NERR_ACFFileIOFail	2229	An error was encountered in accessing the accounts database.
NERR_ACFTooManyLists	2230	Too many lists were specified.
NERR_UserLogon	2231	Deleting a user with a session is not allowed.
NERR_ACFNoParent	2232	The parent directory cannot be located.
NERR_CanNotGrowSegment	2233	It is not possible to enlarge the UAS cache segment.
NERR_SpeGroupOp	2234	This operation is not allowed on this special group.
NERR_NotInCache	2235	This user is not cached in UAS sess cache.
NERR_UserInGroup	2236	The user already belongs to this group.
NERR_UserNotInGroup	2237	The user does not belong to this group.
NERR_AccountUndefined	2238	The user account is undefined.
NERR_AccountExpired	2239	The user account has expired.
NERR_InvalidRequester	2240	The user is not allowed to log on from this requester.
NERR_InvalidLogonHours	2241	The user is not allowed to log on at this time.
NERR_PasswordExpired	2242	The password has expired.
NERR_PasswordCantChange	2243	This password cannot change.
NERR_PasswordHistConflict	2244	This password cannot be used now.
NERR_PasswordTooShort	2245	The password is shorter than required.
NERR_PasswordTooRecent	2246	The password is too recent to change.
NERR_InvalidDatabase	2247	The UAS database file is damaged.
NERR_DatabaseUpToDate	2248	No updates are necessary to this replicant.
NERR_SyncRequired	2249	This user accounts database is outdated because it no longer is synchronized with the domain controller.

Use

A LAN Server API function returning error codes 2250-2252 encountered an error while trying to retrieve information about a resource or while using a resource.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_UseNotFound	2250	The connection cannot be found.
NERR_BadAsgType	2251	This asg_type is not valid.
NERR_DeviceIsShared	2252	This device is already being shared.

Message

A LAN Server API function returning error codes 2270-2300 encountered an error while processing information about a message.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NoComputerName	2270	A computer name has not been configured.
NERR_MsgAlreadyStarted	2271	This message server has already been started.
NERR_MsgInitFailed	2272	The message server initialization request has failed.
NERR_NameNotFound	2273	The message alias cannot be found on the local area network.
NERR_AlreadyForwarded	2274	This message alias has already been forwarded.
NERR_AddForwarded	2275	This message alias has been added, but is still forwarded.
NERR_AlreadyExists	2276	This message alias already exists locally.
NERR_TooManyNames	2277	The maximum number of added message aliases has been exceeded.
NERR_DelComputerName	2278	The computer name cannot be deleted.
NERR_LocalForward	2279	Messages cannot be forwarded back to the same workstation.
NERR_GrpMsgProcessor	2280	Error in the domain message processor.
NERR_PausedRemote	2281	The message has been sent, but the reception is currently paused.

NERR_BadReceive	2282	The message was sent but not received.
NERR_NameInUse	2283	The message alias is currently in use-try again later.
NERR_MsgNotStarted	2284	The Messenger service has not been started.
NERR_NotLocalName	2285	The name is not on the local computer.
NERR_NoForwardName	2286	The forwarded message alias cannot be found on the network.
NERR_RemoteFull	2287	The message alias table on the remote station is full.
NERR_NameNotForwarded	2288	Messages for this alias are not currently forwarded.
NERR_TruncatedBroadcast	2289	The broadcast message was truncated.
NERR_FileError	2290	An error occurred in reading the message file.
NERR_InvalidDevice	2294	This is not a valid device.
NERR_WriteFault	2295	A write fault has occurred.
NERR_DuplicateName	2297	A duplicate message alias exists on the local area network.
NERR_DeleteLater	2298	This message alias will be deleted later.
NERR_IncompleteDel	2299	The message alias was not successfully deleted from all networks.
NERR_MultipleNets	2300	This operation is not supported on machines with multiple networks.

Directory Limits

Return codes 2301-2309 indicate an error with a directory limits process.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_DASDNotInstalled	2301	Directory limits are not enabled on the specified drive.
NERR_DASDAlreadyInstalled	2302	Directory limits are already enabled on the specified drive.
NERR_NoHPFSVolume	2303	The directory is not on a 386-HPFS volume.
NERR_DASDMaxValidationFailed	2304	The supplied directory limit is

		smaller than the current directory size. The limit was not applied.
NERR_DASDInstallVolumeLocked	2305	Support for directory limits on the specified drive cannot complete. The drive is in use or locked by another process. You must shut down and restart your system in order for directory limits to become operational.
NERR_LimitNotFound	2306	The directory limit was not found.
NERR_LimitExists	2307	The directory limit already exists.
NERR_DASDNotRunning	2308	The 386-HPFS file system failed to enable directory limits on the specified drive.
NERR_DASDNotOperational	2309	The NET DASD operation failed. Support for directory limits is enabled, but is not yet operational.
		Cause: Directory limits operations on this drive cannot be performed until support for directory limits is operational.

Server

A LAN Server API return codes 2310-2320 indicate an error with processing information about a server.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NetNameNotFound	2310	This shared resource does not exist.
NERR_DeviceNotShared	2311	This device is not shared.
NERR_ClientNameNotFound	2312	A session does not exist with that computer name.
NERR_FileIdNotFound	2314	There is not an open file with that ID number.
NERR_ExecFailure	2315	A failure occurred when executing a remote administration command.
NERR_TmpFile	2316	A failure occurred when opening a remote temporary file.
NERR_TooMuchData	2317	The data returned from a remote administration command has been truncated to 64KB.
NERR_DeviceShareConflict	2318	This device cannot be shared as both a spooled and a non-spooled device.

NERR_BrowserTableIncomplete	2319	The server table was initialized incorrectly.
NERR_NotLocalDomain	2320	This domain is not active on this computer.

Serial Device

A LAN Server API function returning error codes 2331-2343 encountered an error with a serial device.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_DevInvalidOpCode	2331	The operation is not valid for this device.
NERR_DevNotFound	2332	This device cannot be shared.
NERR_DevNotOpen	2333	This device was not open.
NERR_BadQueueDevString	2334	This device name string is not valid.
NERR_BadQueuePriority	2335	The queue priority is not valid.
NERR_NoCommDevs	2337	There are no shared communication devices.
NERR_QueueNotFound	2338	A queue does not exist for this request.
NERR_BadDevString	2340	This list of devices is not valid.
NERR_BadDev	2341	The requested device is not valid.
NERR_InUseBySpooler	2342	This device is already in use by the spooler.
NERR_CommDevInUse	2343	This device is in use as a serial device.

Input and Output (I/O)

A LAN Server API function returning error codes 2351-2362 encountered an error while processing input or output.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_InvalidComputer	2351	The specified computer name is not valid.
NERR_MaxLenExceeded	2354	The string and prefix specified are too long.
NERR_BadComponent	2356	This path component is not valid.

NERR_CantType	2357	The type of input cannot be determined.
NERR_TooManyEntries	2362	The buffer for types is not big enough.

Audit Log and Error Log

A LAN Server API function returning error codes 2377-2379 encountered an error writing to or reading from the audit log file or error log file.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_LogOverflow	2377	This log file exceeds the maximum defined size.
NERR_LogFileChanged	2378	This log file has changed between reads.
NERR_LogFileCorrupt	2379	This log file is damaged.

Remote Error

A LAN Server API function returning error codes 2380-2392 encountered an error while running a remote process.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_SourceIsDir	2380	The source path cannot be a directory.
NERR_BadSource	2381	The source path is not valid.
NERR_BadDest	2382	The destination path is not valid.
NERR_DifferentServers	2383	The source and destination paths are on different servers.
NERR_RunSrvPaused	2385	The run server you requested using the NET RUN command is paused.
NERR_ErrCommRunSrv	2389	An error occurred when communicating with a run server.
NERR_ErrorExecingGhost	2391	An error occurred when starting a background process.
NERR_ShareNotFound	2392	The shared resource you are connected to could not be found.

Requester and Redirector

A LAN Server API function returning error codes 2400-2406 encountered an error with the Requester and Redirector.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_InvalidLana	2400	The LAN adapter number is not valid.
NERR_OpenFiles	2401	There are open files on the connection.
NERR_ActiveConns	2402	Active connections still exist.
NERR_BadPasswordCore	2403	This netname or password is not valid.
NERR_DevInUse	2404	The device is being accessed by an active process.
NERR_LocalDrive	2405	The drive letter is in use locally.
NERR_PausedConns	2406	Paused connections could not be deleted.

Alert

A LAN Server API function returning error codes 2430-2440 encountered an error while processing the Alerter service.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_AlertExists	2430	The specified client is already registered for the specified event.
NERR_TooManyAlerts	2431	The Alerter service table is full.
NERR_NoSuchAlert	2432	An incorrect or nonexistent alert name was raised.
NERR_BadRecipient	2433	The Alerter service recipient is not valid.
NERR_InvalidLogSeek	2440	The log file does not contain the requested record number.

User Accounts Subsystem and Netlogon

A LAN Server API function returning error codes 2450-2458 encountered an error with the User Accounts Subsystem (UAS) or the Netlogon service.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_BadUasConfig	2450	The UAS is not configured correctly.
NERR_NetLogonNotStarted	2455	The Netlogon service has not been

started.

NERR_CanNotGrowUASFile	2456	It is not possible to enlarge the UAS file.
NERR_PasswordMismatch	2458	A password mismatch has been detected.

Server Integration

A LAN Server API function returning error codes 2460-2467 encountered an error while processing the Server service.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NoSuchServer	2460	The server ID does not specify a valid server.
NERR_NoSuchSession	2461	The session ID does not specify a valid session.
NERR_NoSuchConnection	2462	The connection ID does not specify a valid connection.
NERR_TooManyServers	2463	There is no space for another entry in the table of available servers.
NERR_TooManySessions	2464	The server has reached the maximum number of sessions it supports.
NERR_TooManyConnections	2465	The server has reached the maximum number of connections it supports.
NERR_TooManyFiles	2466	The server cannot open more files because it has reached its maximum number.
NERR_NoAlternateServers	2467	There are no additional servers registered on this server.

Uninterrupted Power Supply

A LAN Server API function returning code 2480 encountered an error while trying to process uninterrupted power supply (UPS) service.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_UPSDriverNotStarted	2480	The UPS driver could not be accessed by the UPS service.

Remoteboot

A LAN Server API function returning codes 2500-2523 encountered an error while processing remote initial program load, or the LAN Server initialized and trademarks are shown.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_BadDosRetCode	2500	The specified program returned an error code.
NERR_ProgNeedsExtraMem	2501	The specified program needs extra memory KB.
NERR_BadDosFunction	2502	The specified program called an unsupported 21H function.
NERR_RemoteBootFailed	2503	Remote IPL is ended.
NERR_BadFileChecksum	2504	The specified file is damaged.
NERR_NoRplBootSystem	2505	There is no system loader for the remote IPL files.
NERR_RplLoadrNetBiosErr	2506	NetBIOS returned an error: the NCB and SMB are dumped above.
NERR_RplLoadrDiskErr	2507	A disk I/O error has occurred.
NERR_ImageParamErr	2508	Image parameter substitution failed.
NERR_TooManyParams	2509	Too many image parameters span disk sectors.
NERR_NonDosFloppyUsed	2510	The image was generated from a DOS format diskette.
NERR_RplBootRestart	2511	Remote IPL will be started again later.
NERR_RplSrvrCallFailed	2512	The call to the remote IPL server failed.
NERR_CantConnectRplSrvr	2513	The requester is unable to connect to the image server.
NERR_CantOpenImageFile	2514	The image file on the remote IPL server cannot be opened.
NERR_CallingRplSrvr	2515	The remote server is being called.
NERR_StartingRplBoot	2516	Remote IPL of the system is starting.
NERR_RplBootServiceTerm	2517	The remote IPL service was ended; check the error log at the remote IPL server for details.
NERR_RplBootStartFailed	2518	Remote IPL startup failed; check the error log for the cause of the problem.
NERR_RPL_CONNECTED	2519	A second connection to a remote IPL resource is not allowed.

NERR_RplBootErrDetected	2520	The remote IPL service detected a configuration error but is continuing. Check the error log at the remote IPL server for details.
NERR_RPL_COPYRIGHT1	2521	Remote IPL Initialization Program Version 4.0.
NERR_RPL_COPYRIGHT2	2522	(C) Copyright International Business Machines Corporation 1989, 1996.
NERR_RPL_COPYRIGHT3	2523	(C) Copyright Microsoft Corporation 1989, 1991.

Fault Tolerance Administration

A LAN Server API function returning error codes 2525-2534 encountered an error while administering Fault Tolerance.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_FTNotInstalled	2525	DISKFT.SYS is not installed. Fault Tolerance cannot be started.
NERR_FTMONITNotRunning	2526	FTMONIT.EXE is not active. Fault Tolerance cannot be started.
NERR_FTDiskNotLocked	2527	An unlock request has been made to a drive that is not locked.
NERR_FTDiskNotAvailable	2528	A request has been made to lock a drive that is already locked by another process.
NERR_FTUnableToStart	2529	The verifier/corrector cannot be started.
NERR_FTNotInProgress	2530	The verifier/corrector cannot be aborted because it is not started.
NERR_FTUnableToAbort	2531	The verifier/corrector cannot be aborted.
NERR_FTUnabletoChange	2532	The disk could not be locked/unlocked.
NERR_FTInvalidErrHandle	2533	The error handle was not recognized.
NERR_FTDriveNotMirrored	2534	The drive is not mirrored.

Alias

A LAN Server API function returning error codes 2781-2788 encountered an error while requesting or changing information about network

resources.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NoAccessDrive	2781	The drive specified is not valid or cannot be accessed.
NERR_AliasExists	2782	The alias already has been created.
NERR_AliasNotFound	2783	The alias does not exist.
NERR_InvAliasDev	2785	The resource specified is not valid.
NERR_DCDBError	2786	The domain control database is unreadable or cannot be accessed at this time.
NERR_NetnameExists	2787	The alias is already used as a spooler queue name.
NERR_DupAliasRes	2788	The resource already is defined by another alias.

Application

A LAN Server API function returning error codes 2792-2796 encountered an error while requesting or modifying information about network applications.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_AppExists	2792	An application with this name already has been created.
NERR_AppNotFound	2793	The application does not exist.
NERR_BadAppRemark	2796	The application remark contains an error or is not unique.

Apply

A LAN Server API function returning error codes 2800-2802 encountered an error while attempting an apply operation on an access control list (ACL).

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_ApplyNotPermitted	2800	The apply operation is not allowed on the specified path.
NERR_IncompleteApply	2801	The apply operation ended prematurely.

NERR_ApplyFailed	2802	An apply error occurred with the following information in the return buffer: Error path: %1 Error code: %2.
------------------	------	--

RIPL

A LAN Server API function returning error codes 5400-5480 encountered an error while requesting or modifying RIPL functions.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_NotRIPLServer	5401	The server name specified is not a valid remote IPL server.
NERR_RPL_MAPNotFound	5402	The RPL.MAP file was not found or could not be opened on the remote IPL server.
NERR_RPL_MAPReadError	5403	An error was detected while reading the RPL.MAP file on the remote IPL server.
NERR_RPL_MAPWriteError	5404	An error occurred while updating the RPL.MAP file on the remote IPL server.
NERR_MachineNotFound	5405	The machine definition does not exist on the remote IPL server.
NERR_FITSFileNotFound	5406	An error was detected while reading configuration files in the IBMLAN\RPL\FITS directory on the remote IPL server.
NERR_MACHINESFileNotFound	5407	An error was detected while reading configuration files in the IBMLAN\RPL\MACHINES directory tree on the remote IPL server.
NERR_IBMCOMFileNotFound	5408	An error was detected while reading configuration files in the IBMLAN\RPL\IBMCOM directory tree on the remote IPL server.
NERR_RPLUSERFileNotFound	5409	An error was detected while reading configuration files in the IBMLAN\RPLUSER directory tree on the remote IPL server.
NERR_RPLFileNotFound	5410	An error was detected while reading configuration files in the IBMLAN\RPL directory tree on the remote IPL server.
NERR_MachineFilesLocked	5411	Machine configuration files are locked by the remote IPL server file system.

NERR_ServerRecordInvalid	5412	The server record identifier is not defined in an enabled server record entry in the RPL.MAP file.
NERR_RemarkInvalid	5413	The machine definition remark was more than 48 bytes.
NERR_NDISProfileNotFound	5414	The network adapter directory profile could not be found in the NDISDD.PRO file on the remote IPL server.
NERR_MachNameTooLong	5415	Machine definition names for DOS or FAT file systems cannot exceed 8 bytes.
NERR_CreateMachNameInvalid	5420	The machine definition could not be created, because the name is not unique in the domain.
NERR_CreateMachNameExists	5421	The machine definition already exists on the remote IPL server.
NERR_CreateMachModelInvalid	5422	The model machine name does not exist on the remote IPL server.
NERR_CreateMachDriveInvalid	5424	The remote IPL OS/2 boot drive identifier is not valid.
NERR_NetworkAddressInvalid	5425	The network adapter address is not valid.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_CreateMachAddrExists	5426	A machine definition already exists with the specified network adapter address.
NERR_CreateMachReadError	5428	An error was detected while reading machine configuration files on the remote IPL server.
NERR_CreateMachWriteError	5429	An error was detected while writing machine configuration files on the remote IPL server.
NERR_CreateMachFileNotFound	5430	Files required for creating a machine definition could not be found on the remote IPL server.
NERR_CreateMachDirFailure	5432	An error was detected while creating machine configuration directories on the remote IPL server.
NERR_CreateMachACLFailure	5433	An error was detected while creating access control profiles on the remote IPL server.
NERR_DelMachFailure	5440	The machine definition could not be deleted.
NERR_DelMachDirFailure	5441	An error was detected while deleting

		machine configuration directories on the remote IPL server.
NERR_EnumMachNotFound	5450	A machine definition of the type requested does not exist on the remote IPL server.
NERR_SetMachDriveFailure	5470	The remote IPL OS/2 boot drive identifier cannot be changed because the machine has a Workplace Shell desktop defined.
NERR_SetMachNameInvalid	5471	The machine definition name cannot be changed.
NERR_SetMachVersionInvalid	5472	The machine definition OS/2 version cannot be changed.
NERR_SetMachCompInvalid	5473	The machine definition parameter specified is not valid for the current machine definition type.
NERR_DOSImageNotFound	5480	The DOS image file name does not exist.

Directory and Security Server

A Directory and Security Server function returning error codes 7500-7900 encountered an error while administering DSS.

SYMBOLIC CONSTANT	VALUE	MEANING
NERR_DCEError	7500	DCE error status returned.
NERR_DCEErrorLogged	7501	DCE error status has been logged.
NERR_NoResDomName	7502	No resource domain name is specified.
NERR_ResDomNotFound	7503	The specified resource domain does not exist.
NERR_InvalidObjGlobalName	7504	The specified object global name is not valid.
NERR_InvalidResDomName	7505	The specified resource domain name is not valid.
NERR_ResDomInvalidParm	7506	One or more of the input parameters is invalid.
NERR_BadUserDataInRgy.	7507	User data in registry is not valid for DSS
NERR_SchemaNotFound	7508	Unable to find DSS required schema entry.
NERR_Consistency_Error	7509	A secondary error occurred while attempting to recover from an error.

		The object is now in an unknown state.
NERR_OrgNotFound	7510	The specified organization/policy group does not exists.
NERR_AuthTicketExpired	7511	The user's DCE authorization ticket has expired.
NERR_RegistryNotFound	7512	Unable to contact the DCE security registry.
NERR_UserNotInResdom	7513	The user is not a member of the specified resource domain.
NERR_UserExistsInCell	7514	The user already exists in the cell..
NERR_GroupNotInResDom	7515	The group is not defined to the specified resource domain.
NERR_GroupExistsInCell	7516	The group already exists in the cell.
NERR_GroupNotInResdom	7517	The group is not defined to the specified resource domain.
NERR_ParentResDomNotFound	7518	The specified parent resource domain does not exist.
NERR_ResDomAlreadyExists	7519	The specified resource domain already exists.
NERR_ResDomNotEmpty	7520	Resource domain of delete request is not empty.
NERR_AppNotInResDom	7521	The application is not in the specified resource domain.
NERR_AliasNotInResDom	7522	The alias is not defined to the specified resource domain.
NERR_PrimaryGroup	7523	User cannot be deleted from their primary group.
NERR_ResDomTooManyGroups	7524	The resource domain contains too many group entries to be a sync resource domain.
NERR_ResDomMaxGroupEntries	7525	The resource domain already contains the maximum number of entries permitted.
NERR_InvalidUserType	7526	Invalid user type specified.
NERR_SpecialGroupAddReject	7527	Can not add special group.
NERR_SpecialGroupDeleteReject	7528	Can not delete special group.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_CellnameConflict	7529	Invalid user type specified.
NERR_InvalidCellName	7530	Invalid cell name syntax.
NERR_InvalidAliasNameForma	7531	The syntax of the alias name is not

valid.

NERR_IncompatibleLevel	7532	The level specified is incompatible for this resource..
NERR_ResDomServerExists	7533	The server object already exists in the resource domain.
NERR_RootResDomDelNotAllowed	7534	The root resource domain of a cell may not be deleted.
NERR_PropertiesMissing	7535	Not all properties of an object were specified.
NERR_PrimaryGroupNotFound	7536	The specified primary group does not exist.
NERR_PropertyDependency	7537	The property definition tied to the following properties.
NERR_WrongDataType	7538	An invalid property data type was specified.
NERR_MultiplePropertyDefs	7539	More than one instance of a property is allowed.
NERR_NoResDomCLI	7540	No resource domain name is specified.
NERR_RsrvdResName	7541	Can not use reserved resource name.
NERR_InvalidResDomSyntax	7542	The syntax of the resource domain is invalid.
NERR_ResDomsLocked	7543	The syntax of the resource domain name is invalid.
NERR_FullSyncCanceled	7544	DIRSYNC full sync request terminated prior to completion.
NERR_UserInResdom	7545	The user is already a member of the specified resource domain.
NERR_NoCDSBind	7546	Can not bind to DCE CDS server.
NERR_DSSDCESTART	7547	Error %1 occurred while running DCESTART.
NERR_DSSDCESTOP	7548	Error %1 occurred while running DCESTOP.
NERR_IncompatibleResom	7553	The domain name does not match the resource domain broadcast name.
NERR_EncryptedPassword	7554	An error occurred while attempting to set the user
NERR_PasswordHistory	7555	An error occurred while attempting to update the user
NERR_DSSRejection	7556	A DSS client attempted to logon to a resource domain.
NERR_RgySyncFail	7557	An error occurred when initializing the rgysync worker process.

NERR_PasswordAllSpaces	7650	The password can not be all spaces.
NERR_PasswodAllAlphNum	7651	The password must contain at least one non-alphanumeric character.
NERR_ServerPasswordInconsi te	7652	The server password in the local keytab file is inconsistent with the server
NERR_ACLDBOpenFail	7750	The ACL databases could not be opened. The DCE error follows.
NERR_RdaclInitFail	7751	The server could not register the rdacl interface to RPC. The DCE error follows.
SYMBOLIC CONSTANT	VALUE	MEANING
NERR_ACLDeleteFail	7752.	The ACL for %1 could not be deleted. The DCE error follows.
NERR_ACLInheritFail	7753	The initial ACL for %1 could not be created. The DCE error follows.
NERR_ACLMoveFail	7754	The ACL for %1 could not be renamed. The DCE error follows.
NERR_MediaDetection	7755	An error %1 occurred while determining drives with removable media.
NERR_ACLReadFail	7756	The ACL for %1 could not be read. The DCE error follows.
NERR_ACLWriteFail	7757	The ACL for %1 could not be updated. The DCE error follows.
NERR_RdaclListenFail	7758	An rpc error occurred, the server is no longer processing sec_acl APIs. The DCE error follows.
NERR_RootAclError	7759	The ACL for %1, the device root is missing.
NERR_CredMgrError	7760	Credential Manager is not started (returned by sinlogon).
NERR_CreateLgcyPac	7761	Unable to create PAC for Legacy Client (returned by CreateLgcyPac).
NERR_CredManagementError	7762.	A DCE error occurred while managing the process credentials. The data is the DCE error.
NERR_KeyManagementError	7763	A DCE error occurred while managing the processes login key. The data is the DCE error.
NERR_PACBufTooSmall	7764	The global buffer for the PAC is too small.
NERR_ExchangeTokens	7765	Unable to exchange DCE security tokens.

NERR_RequestToken	7766	Unable to request DCE security token.
NERR_ReleaseBuffer	7767	Unable to release memory associated with a DCE buffer.
NERR_RequestInfo	7768	Request for DCE information failed.
NERR_MountableMedia	7774	The operation is not valid on a removable media device.
NERR_ApplyUnexpectedError	7799	The apply server encountered an unexpected error.

User Profile Management

A UPM function returning the following error codes encountered an error.

SYMBOLIC CONSTANT	MEANING
UPM_ACTIVE	The domain logon or logoff failed; a domain logon, logoff, or the LAN Requester graphical user interface is active.
UPM_BAD_PARAMETER	The user ID or remote name was not valid.
UPM_BAD_TYPE	The remote type specified was not valid.
UPM_ERROR_MORE_DATA	All the available UPM_USER_LOGON entries did not fit into buf.
UPM_FAIL_SECURITY	The user ID or password could not be validated.
UPM_LOG_CANCEL	The user cancelled from the logon panel.
UPM_LOG_FILE_NOT_FOUND	A required logon program was not found.
UPM_LOG_INPROC	Another logon was being processed; the logon attempt was not successful.
UPM_LOGGED	The domain logon failed because a domain logon has already occurred.
UPM_NOMEM	Insufficient memory is available to process the request.
UPM_NOT_LOGGED	The user has not logged on locally.
UPM_OK	The user was logged on successfully.
UPM_OPEN_SESSIONS	The domain logoff failed because a

	domain logon has active open sessions.
UPM_PASSWORD_EXP	The logon failed; the user's password is expired.
UPM_SS_PWDEXPWARNING	The local logon succeeded. The user's password is expired.
UPM_SS_BUSY	The local logon failed; the secure shell was busy.
UPM_SS_DEAD	The local logon failed; the secure shell has stopped.
UPM_SYS_ERROR	A system error occurred.
UPM_ULP_LOADED	The local logon failed; a local logon with a user logon profile is already active.
UPM_UNAVAIL	The logon failed; the remote node or domain could not be contacted to process the logon request.

REXX RIPL Function Calls

This appendix includes the following REXX functions for RIPL workstation support:

[REXX - RxNetCreateRIPLMachine](#)
[REXX - RxNetDeleteRIPLMachine](#)
[REXX - RxNetEnumRIPLMachine](#)
[REXX - RxNetGetRIPLMachineInfo](#)
[REXX - RxNetSetRIPLMachineInfo](#)

The APIs in this category provide a REXX interface that is similar to the RIPL APIs described in [RIPL Category](#). The REXX functions reside in a DLL provided by LAN Server and are addressable from within a REXX program.

From within REXX, you also can create an addressable register function that can be called from an outside C application. This is set up from the REXX side first, by calling a routine from within a REXX function call. For example:

```
call RxFuncAdd 'RxRegRIPLFuncs', 'RXRPLEXT', 'RXREGRIPLFUNCS'
call RxRegRIPLFuncs
```

This technique can be used to make all the REXX RIPL functions addressable.

The amount of information returned depends on the value of the *Level* parameter passed with each function. When *Level* is 0 (workstation name only), each line contains the name of a single workstation. A level of 1 returns a line for each workstation beginning with the workstation name. This is followed by the delimiter pattern, `^' (*space* ^ *space*), followed by the **Remark** field. A similar format is used for levels 2 and 12. For example, a return from the *RxNetEnumRIPLMachine* function at level 0 might be:

```
Machine1
Machine2
```

while the output at level 2 would appear as:

```
Machine1 ^ remarks for 1 ^ abc123456789 f DOSIMAGE ^
          DOS_SERV_REC_ID ^ OS2_SERV_REC_ID
```


Notice that only variable-length fields are separated by delimiters.

RxNetCreateRIPLMachine

RxNetCreateRIPLMachine

The RxNetCreateRIPLMachine API creates the RIPL definition for a requester.

Restrictions

This function requires either Administrator authority or Server operator privilege. If a session already is active between a workstation and its RIPL server, this API does not function at level 12.

Syntax

```
RxNetCreateRIPLMachine ServerName , Level ,  
                        VariableArguments
```

Parameters

<i>ServerName</i>	is the name of the RIPL server on which the function is to run.
<i>Level</i>	must be either 2 or 12, specifying the level of information to return. (See RIPL Machine Information Level 2 and RIPL Machine Information Level 12 for a discussion of the type of information returned by this function.)
<i>VariableArguments</i>	is a variable-length list of parameters. (The parameters that are passed depend on which level is used.)

Examples

```
call RxNetCreateRIPLMachine 'ripl_server' , '2' , 'ripl_requester' ,  
                             'remarks...' , '123456789abc' , 'e' , 'imageid' ,  
                             'DOS_RECORD' , 'OS2_RECORD'  
  
call RxNetCreateRIPLMachine 'ripl_server' , '12' , 'ripl_requester' ,  
                             'remarks...' , '123456789abc' , 'ripl_model'
```

Related Information

For more information, see [RIPL - NetCreateRIPLMachine](#).

RxNetDeleteRIPLMachine

RxNetDeleteRIPLMachine

The RxNetDeleteRIPLMachine API deletes a RIPL requester definition.

Restrictions

This function requires either Administrator authority or Server operator privilege.

Syntax

```
RxNetDeleteRIPLMachine ServerName , MachineName
```

Parameters

<i>ServerName</i>	is the name of the RIPL server on which the function is to run.
<i>MachineName</i>	is the name of the workstation definition that is to be deleted.

Example

```
call RxNetDeleteRIPLMachine 'ripl_server' , 'ripl_requester'
```

Related Information

For more information, see [RIPL - NetDeleteRIPLMachine](#).

RxNetEnumRIPLMachine

RxNetEnumRIPLMachine

The RxNetEnumRIPLMachine API lists RIPL requesters of the specified types on a server.

Restrictions

User authority is allowed limited access to this function.

Syntax

```
RxNetEnumRIPLMachine ServerName , Level ,  
                     Type , RtnVariable
```

Parameters

<i>ServerName</i>	is the name of the RIPL server on which the function is to run.
<i>Level</i>	must be 0, 1, 2, or 12, specifying the amount of information needed.
<i>Type</i>	indicates the type of requester to enumerate. Specify a value of 1 for an OS/2 RIPL requester or a value of 2 for a DLS RIPL requester.
<i>RtnVariable</i>	is the stem of a variable that holds the output.

Example

```
call RxNetEnumRIPLMachine 'ripl_server' , '0' , '1' , 'variable_name'
```

Remarks

Access authority is not required to call this function at level 0 or level 1. For level 2 and 12, the user must have Administrator authority or Server operator privilege.

Related Information

For more information, see [RIPL - NetEnumRIPLMachine](#).

RxNetGetRIPLMachineInfo

RxNetGetRIPLMachineInfo

The RxNetGetRIPLMachineInfo API retrieves information about a RIPL requester.

Restrictions

User authority is allowed limited access to this function. A user with any privilege level can call this function with a level of 0 or 1. To call the function with a level of 2, the user must have Administrator authority or server operator rights.

Syntax

```
RxNetGetRIPLMachineInfo ServerName , Level ,  
                        MachineName , RtnVariable
```

Parameters

<i>ServerName</i>	is the name of the RIPL server on which the function is to run.
<i>Level</i>	must be 0, 1, or 2, specifying the amount of information needed.
<i>MachineName</i>	is the name of the workstation for which information is to be retrieved.
<i>RtnVariable</i>	is the stem of a variable that holds the output.

Example

```
call RxNetGetRIPLMachineInfo 'ripl_server' , '0' ,  
    'ripl_requester2' , 'variable_name'
```

Related Information

For more information, see [RIPL - NetGetRIPLMachineInfo](#).

RxNetSetRIPLMachineInfo

RxNetSetRIPLMachineInfo

The RxNetSetRIPLMachineInfo API changes information about a RIPL requester.

Restrictions

Administrator authority is required for this API except when a user issues a call concerning the user

This function requires either Administrator authority or Server operator privilege.

Syntax

```
RxNetSetRIPLMachineInfo ServerName , Level , ParmNum ,  
                        MachineName , VariableArguments
```

Parameters

<i>ServerName</i>	is the name of the RIPL server on which the function is to run.
<i>Level</i>	must be 1 or 2, specifying the amount of information needed.
<i>ParmNum</i>	must be 0, 2, 3, 5, 7, 9, or 10. Except for 0, these values correspond to the fields in the RPL.MAP file and indicate which parameter is to be returned in <i>VariableArguments</i> . When <i>ParmNum</i> is 0, all information in either the <i>ripl_machine_info_1</i> (RIPL Machine Information Level 1) or <i>ripl_machine_info_2</i> (RIPL Machine Information Level 2) data structure is returned, depending on the value of <i>Level</i> .
<i>MachineName</i>	is the name of the workstation for which information is to be changed.
<i>VariableArguments</i>	is a variable-length list of parameters. The parameters that are passed depend on the level at which the function is called.

Examples

```
call RxNetSetRIPLMachineInfo 'ripl_server' , '1' , '2' ,  
                             'ripl_requester' , 'remarks...'  
  
call RxNetSetRIPLMachineInfo 'ripl_server' , '2' , '0' ,  
                             'ripl_requester' , 'remarks...' , '123456789abc' ,  
                             'e' , 'imageid' , 'DOS_RECORD' , 'OS2_RECORD'
```

Related Information

For more information, see [RIPL - NetSetRIPLMachineInfo](#).

Notes for DOS Applications

This appendix lists for each LAN Server API category the differences between calls from DLS and OS/2 workstations. It also provides a table of the header files used under DOS.

Differences in API Functions for DOS

DOS LAN Services (DLS) supports a subset of the API functions supported by an OS/2 workstation. Most of the DOS LAN Services API functions are processed on a remote server. API functions that can be processed remotely must include a remote server name parameter to identify where the function is to be run. Attempting to run a remote-only function on a local requester returns NERR_RemoteOnly.

Those functions that can be processed on a local requester must be called with either a NULL server name parameter (defaults to local requester name) or the name assigned the local requester in the format of *computername*. Attempting to run a local-only function on a remote server returns ERROR_NOT_SUPPORTED.

The following sections describe the differences between a call from a DLS and an OS/2 workstation.

Access Permission

The functions in the access permission category examine or change user or group access permission records for server resources. Under DOS, these functions can be processed only on a remote server. Administrator authority is required to use the functions. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetAccessAdd	Remote only
NetAccessDel	Remote only
NetAccessEnum	Remote only
NetAccessGetInfo	Remote only
NetAccessGetUserPerms	Remote only
NetAccessSetInfo	Remote only

Auditing

The functions in the auditing category control the audit log file, which contains an audit trail of operations that occur on a server. Under DOS, these functions can be processed only on a remote server. Administrator authority is required to use the functions. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetAuditClear	Remote only
NetAuditRead	Remote only
NetAuditWrite	Not supported

Configuration

Configuration APIs can be called locally on a DLS requester to retrieve information from the local NETWORK.INI file. These APIs can be called remotely from a DLS requester to retrieve information from a remote IBMLAN.INI file.

FUNCTION	DIFFERENCES FOR DOS
NetConfigGet2	None
NetConfigGetAll2	None

Connection

The NetConnectionEnum function lists all connections made either to a server by a requester or to a shared resource of a server. Under DOS, these functions can be processed only on a remote server. Administrative privilege is required to use the functions. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetConnectionEnum	Remote only

Error Logging

The functions in the error logging category control the error log file.

FUNCTION	DIFFERENCES FOR DOS
NetErrorLogClear	Remote only
NetErrorLogRead	Remote only

File

The functions in the file category provide a system for (1) monitoring the file, device, and pipe resources that are opened on a server, and (2) closing these resources if necessary. Under DOS, these functions can be processed only on a remote server. Administrator authority is required to use the functions. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetFileClose2	Remote only
NetFileEnum2	Remote only
NetFileGetInfo2	Remote only

Group

The functions in the group category control user groups in the user accounts subsystem (UAS) database. Under DOS, these functions can be processed only on a remote server. Administrator authority is required to use the functions. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
----------	---------------------

NetGroupAdd	Remote only
NetGroupAddUser	Remote only
NetGroupDel	Remote only
NetGroupDelUser	Remote only
NetGroupEnum	Remote only
NetGroupGetInfo	Remote only
NetGroupGetUsers	Remote only
NetGroupSetInfo	Remote only
NetGroupSetUsers	Remote only

Handle

The functions in the handle category call and set information on a per-handle basis.

FUNCTION	DIFFERENCES FOR DOS
NetHandleGetInfo	Can be called only at level 1 and only with remote named pipes
NetHandleSetInfo	Can be called only at level 1 and only with remote named pipes

HPFS386 Information and Administration

The functions in the HPFS386 information and administration category differ for DLS as follows:

FUNCTION	DIFFERENCES FOR DOS
NetDASDAdd	Remote only
NetDASDDel	Remote only
NetDASDGetInfo	Remote only
NetDASDSetInfo	Remote only
NetDASDEnum	Remote only
NetDASDCheck	Remote only
NetDASDctl	Remote only

Mailslot

The functions in the mailslot category provide one-way interprocess communication (IPC). Under DOS, the functions can be processed on a local requester or remote server.

Mailslots can be read or deleted only by the process that created them. Mailslots created by a process are deleted when that process ends.

FUNCTION	DIFFERENCES FOR DOS
DosDeleteMailslot	None
DosMailslotInfo	None
DosMakeMailslot	None
DosPeekMailslot	None
DosReadMailslot	None
DosWriteMailslot	None

Message

The functions in the message category send and receive messages. Under DOS, messages cannot be forwarded or unforwarded.

DLS accepts only the names of users or requesters that are registered in the message name table.

The maximum size of a message under DOS is 64KB.

FUNCTION	DIFFERENCES FOR DOS
NetMessageBufferSend	None
NetMessageFileSend	
NetMessageLogFileGet	None
NetMessageLogFileSet	None
NetMessageNameAdd	None
NetMessageNameDel	None
NetMessageNameEnum	None
NetMessageNameFwd	Remote only
NetMessageNameGetInfo	None
NetMessageNameUnFwd	Remote only

Named Pipe

The functions in the named pipe category control interprocess communication (IPC) for named pipes. The functions can be processed only on a remote server that has interprocess communication shares.

DOS supports only client processes; a pipe already must have been created and connected to on a remote server. Child processes inherit the open file handles of the parent processes. DOS supports asynchronous reading and writing of named pipes, as long as the pipe already is created and connected.

DosReadAsyncNmPipe and DosWriteAsyncNmPipe are DOS named pipe APIs that are not used by the OS/2 program. DosReadAsync is the OS/2 equivalent of DosReadAsyncNmPipe, and DosWriteAsync is the OS/2 equivalent of DosWriteAsyncNmPipe. These companion APIs perform the same functions.

Note: The FAPI replacement library routine for DosOpen provides support for DASD opens (open mode flag 0x8000). Since DOS does not support this operation, pipe operations on this type of file handle return ERROR_INVALID_HANDLE, rather than ERROR_BAD_PIPE.

FUNCTION	DIFFERENCES FOR DOS
DosBufReset	Returns ERROR_BROKEN_PIPE, if the handle is to a named pipe that already has been closed.
DosCallNmPipe	None
DosClose	None
DosDupHandle	None
DosOpen	None
DosPeekNmPipe	None
DosQFHandState	None
DosQHandType	None
DosQNmPHandState	None
DosQNmPipeInfo	None
DosRead	None
DosReadAsyncNmPipe (DOS only)	<p>This is a DOS-unique API.</p> <p>The second DosReadAsyncNmPipe parameter points to Async Notification Routine (pANR); in DosReadAsync, it points to RamSemaphore. Although their names are different, both pANR and RamSemaphore are post-routine addresses used to signal the caller that the read operation is complete.</p> <p>NOTE: If the ERROR_NOT_ENOUGH_MEMORY code is returned, either reduce the size of the cbBuf parameter, or increase the extra heap parameter in the NETWORK.INI file.</p> <p>For an example of a call to DosReadAsyncNmPipe, see Examples.</p>

DosSetFHandState	None
DosSetNmPHandState	None
DosTransactNmPipe	None
DosWaitNmPipe	None
DosWrite	None
DosWriteAsyncNmPipe (DOS only)	<p>This is a DOS-unique API.</p> <p>The second DosWriteAsyncNmPipe parameter points to Async Notification Routine (pANR); in DosWriteAsync, it points to RamSemaphore. Although their names are different, both pANR and RamSemaphore are post-routine addresses used to signal the caller that the write operation is complete.</p> <p>NOTE: If the ERROR_NOT_ENOUGH_MEMORY code is returned, either reduce the size of the cbBuf parameter, or increase the extra heap parameter in the NETWORK.INI file.</p>

Examples

DosReadAsyncNmPipe does an asynchronous read from a specified file; DosWriteAsyncNmPipe does an asynchronous write from a specified file. Each API transfers the specified number of bytes from a file to a buffer, asynchronously with the requesting process execution. These APIs are exclusive to DOS.

```

USHORT APIENTRY DosReadAsyncNmPipe (
    HFILE hf,                /* file handle */
    PULONG Pfunc,            /* address of Postroutine */
    PUSHORT pusErrCode,      /* signals end of read */
    PVOID pBuf,              /* return code (returned) */
    USHORT cbBuf,            /* input buffer */
    USHORT cbBuf,            /* number of bytes to be read */
    PUSHORT pcbBytesRead);   /* number of bytes read */

USHORT APIENTRY DosWriteAsyncNmPipe (
    HFILE hf,                /* file handle */
    PULONG Pfunc,            /* address of Post routine */
    PUSHORT pusErrCode,      /* indicates end of write */
    PVOID pBuf,              /* return code (returned) */
    USHORT cbBuf,            /* output buffer */
    USHORT cbBuf,            /* number of bytes to write */
    PUSHORT pcbBytesWritten); /* number of bytes written */

unsigned far pascal AsyncCB(char far *);
int complete = 0;

main()
{
    HFILE    open_pipe_handle; /* used by DosOpen to open pipe */
    PVOID    buf;
    USHORT   buf_length;
    USHORT   bytes_written, bytes_read;
    USHORT   pipe_async_return_code;

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    /** DosWriteAsyncNmPipe - Asynchronous write to file

```

```

*
*   Writes to a file or a pipe asynchronously with respect
*   to the requesting process execution.
*   (for DOS applications)
*/
complete = 0;
ret = DosWriteAsyncNmPipe((HFILE)open_pipe_handle, /* Write Async Name */
                          /* Pipe call */
                          AsyncCB,
                          (USHORT)&pipe_async_return_code,
                          (PVOID)buf,
                          (USHORT)buf_length,
                          (USHORT)bytes_written);
while (complete == 0);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/** DosReadAsyncNmPipe - Asynchronous read to file
*
*   Reads from a file or a pipe asynchronously with respect
*   to the requesting process's execution.
*   (for DOS applications)
*/
complete = 0;
ret = DosReadAsyncNmPipe((HFILE)open_pipe_handle,
                          /* Read Async Named Pipe call */
                          AsyncCB,
                          (USHORT)&pipe_async_return_code,
                          (PVOID)buf,
                          (USHORT)buf_length,
                          (USHORT)&bytes_read);
while (complete == 0);
exit(0);
}

unsigned far pascal AsyncCB(buffer)      /* exit routine */
char far * buffer;
{
  complete = 1;
  return;
}

```

The following is an example of a compiler statement for DosAsyncNamedPipe APIs:

```
cl /c /Lc /W2 /Zep /Alfu /Gs /Gt10 %1 /I ..\INC /I c:\MYBLD\INC
```

where %1 is the .C file being compiled.

The following is an example of a link statement for DosAsyncNamedPipe APIs:

```
link %1,,mlibcer c:\MYLIB\DOSNET.LIB c:\MYLIB\ c:\MYLIB\API.LIB
```

where %1 is the .OBJ being linked.

Print Destination

The functions in the print destination category control the printers that receive spooled print jobs on a server for 16-bit applications.

FUNCTION	DIFFERENCES FOR DOS
DosPrintDestAdd	Remote only
DosPrintDestControl	Remote only
DosPrintDestDel	Remote only

DosPrintDestEnum	Remote only
DosPrintDestGetInfo	Remote only
DosPrintDestSetInfo	Remote only

Print Job

The functions in the print job category control the print jobs in a printer queue on a server for 16-bit applications.

FUNCTION	DIFFERENCES FOR DOS
DosPrintJobContinue	Remote only
DosPrintJobDel	Remote only
DosPrintJobEnum	Remote only
DosPrintJobGetId	Remote only
DosPrintJobGetInfo	Remote only
DosPrintJobPause	Remote only
DosPrintJobSetInfo	Remote only

Print Queue

The functions in the print queue category control the printer queues on a server for 16-bit applications.

FUNCTION	DIFFERENCES FOR DOS
DosPrintQAdd	Remote only
DosPrintQContinue	Remote only
DosPrintQDel	Remote only
DosPrintQEnum	Remote only
DosPrintQGetInfo	Remote only
DosPrintQPause	Remote only
DosPrintQPurge	Remote only
DosPrintQSetInfo	Remote only

Remote Utility

The functions in the remote utility category enable applications to (1) copy and move remote files, and (2) access the time-of-day information on a remote server. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetRemoteCopy	None
NetRemoteMove	None
NetRemoteTOD	None

Requester

The functions in the requester category control the operation of requesters.

FUNCTION	DIFFERENCES FOR DOS
NetWkstaGetInfo	None
NetWkstaSetInfo	wkix_oth_domains component cannot be set under DOS where x can be either 1 or 10.
NetWkstaSetUID2	Local only

The following table shows the settable components of the *wksta_info_x* data structure and whether a component is used by the DLS.

PARAMETER	USED
charwait	YES
chartime	YES
charcount	YES
errlogsz	NO
printbuftime	NO
wrkheuristics	YES

Serial Device

The functions in the serial device category control shared serial devices and their associated queues.

FUNCTION	DIFFERENCES FOR DOS
NetCharDevControl	Remote only
NetCharDevEnum	Remote only
NetCharDevGetInfo	Remote only
NetCharDevQEnum	Remote only
NetCharDevQGetInfo	Remote only
NetCharDevQPurge	Remote only
NetCharDevQPurgeSelf	Remote only
NetCharDevQSetInfo	Remote only

Server

The functions in the server category enable remote administration tasks to be performed on a remote server. NetServerEnum2 can be processed on either a local requester or a remote server; all other server functions are run on a remote server. Attempting to run NetServerAdminCommand or NetServerGetInfo on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetServerAdminCommand	None
NetServerDiskEnum	Usually called locally (same as the OS/2 program)
NetServerEnum2	None
NetServerGetInfo	None
NetServerSetInfo	None

Service

The functions in the service category control network service programs. They are run on a local requester. Attempting to process the functions on a remote server returns ERROR_NOT_SUPPORTED.

Under DOS, the services cannot be completely stopped; they can be paused and continued, however, with NetServiceControl.

FUNCTION	DIFFERENCES FOR DOS
NetServiceControl	None
NetServiceEnum	None

NetServiceGetInfo	None
NetServiceInstall	Remote only

Session

The functions in the sessions category control network sessions established between requesters and servers.

FUNCTION	DIFFERENCES FOR DOS
NetSessionDel	None
NetSessionEnum	None
NetSessionGetInfo	None

Share

The functions in the share category control shared resources. They can be processed only on a remote server. Attempting to run the functions on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetShareAdd	Remote, or local if Peer Service is installed
NetShareCheck	Remote only
NetShareDel	Remote, or local if Peer Service is installed
NetShareEnum	Remote, or local if Peer Service is installed
NetShareGetInfo	Remote only
NetShareSetInfo	Remote only

Statistics

The functions in the statistics category retrieve and clear the operating statistics for requesters and servers.

FUNCTION	DIFFERENCES FOR DOS
NetStatisticsGet2	None

Use

The functions in the use category examine or control connections (uses) between requesters and servers.

Note: DOS users cannot redirect serial devices with the NET USE command or the NetUseAdd API.

FUNCTION	DIFFERENCES FOR DOS
NetUseAdd	None
NetUseDel	None
NetUseEnum	None
NetUseGetInfo	None

User

The NetUserPasswordSet function controls a user's password account on a server. NetUserPasswordSet can be processed only on a remote server running user-level security. Attempting to run it on a local requester returns NERR_RemoteOnly.

FUNCTION	DIFFERENCES FOR DOS
NetUserAdd	Remote only
NetUserDel	Remote only
NetUserEnum	Remote only
NetUserGetGroups	Remote only
NetUserGetInfo	Remote only
NetUserModalsGet	Remote only
NetUserModalsSet	Remote only
NetUserPasswordSet	Remote only
NetUserSetGroups	Remote only
NetUserSetInfo	Remote only

Table of Header Files

The following header files are used under DOS. NETCONS.H and NETERR.H are not used in every category and therefore are not listed here.

API CATEGORY	HEADER FILE	API CATEGORY	HEADER FILE
Access permis- sion	ACCESS.H	Requester	WKSTA.H
Auditing	AUDIT.H		ACCESS.H
Configuration	CONFIG.H	Serial device	CHARDEV.H
Error logging	ERRLOG.H	Server	SERVER.H
File	SHARES.H	Service	SERVICE.H
Group	ACCESS.H	Session	SHARES.H
Handle	CHARDEV.H	Share	SHARES.H
Mailslot	MAILSLOT.H		ACCESS.H
Message	MESSAGE.H	Statistics	NETSTATS.H
Named pipe	NMPIPE.H	Use	USE.H
Print/Spooler	DOSPRINT.H	User	ACCESS.H
Remote utility	REMUTIL.H		

LAN API Symbolic Constants

This appendix lists the symbolic constants associated with variable-length ASCIIZ strings to which data structure components used in the OS/2 LAN APIs point.

The items listed under the **Component** column are either offsets or pointers to variable-length ASCIIZ strings. In most instances, these ASCIIZ strings can be from 0 to some maximum number of bytes long. The maximum length for the variable-length ASCIIZ strings is a *symbolic constant* that is defined with a value in the NETCONS.H header file.

The variable-length ASCIIZ string should not be greater in length than its symbolic constant + 1 byte. The +1 is to allow for the ending NULL character of the string.

The **Symbolic Constant** column lists the symbolic constant defined in the NETCONS.H header file.

The **Component** column lists the data structure components that act as a pointer or are offset to a variable-length ASCIIZ string.

The **Data Structure** column lists the data structure of which the component is a member.

The **Category** column lists the category to which the data structure belongs.

SYMBOLIC CONSTANT	COMPONENT	DATA STRUCTURE	CATEGORY
CNLEN	computername	print_other_info	Alert
	computername	user_other_info	Alert
	ai2_server	alias_info_2	Alias

	ae_so_compname	ae_sesslogon	Auditing
	ae_sf_compname	ae_sesslogoff	Auditing
	ae_sp_compname	ae_sesspwerr	Auditing
	ae_ct_compname	ae_connstart	Auditing
	ae_cp_compname	ae_connstop	Auditing
	ae_cr_compname	ae_connrej	Auditing
	ae_ra_compname	ae_resaccess	Auditing
	ae_rr_compname	ae_resaccessrej	Auditing
	ae_cf_compname	ae_closefile	Auditing
	ae_am_compname	ae_aclmod	Auditing
	ae_um_compname	ae_uasmod	Auditing
	ae_no_compname	ae_netlogon	Auditing
	ae_nd_compname	ae_netlogondenied	Auditing
	ae_al_compname	ae_acclim	Auditing
	conil_netname	connection_info_1	Connection
	wki0_computername	wksta_info_0	Requester
	wki0_logon_server	wksta_info_0	Requester
	wkil_computername	wksta_info_1	Requester
	wkil_logon_server	wksta_info_1	Requester
	wkil0_computername	wksta_info_10	Requester
	sesil_cname	session_info_1	Session
	sesi2_cname	session_info_2	Session
	sesi10_cname	session_info_10	Session
	usri2_logon_server	user_info_2	User
	usril1_logon_server	user_info_11	User
	usrreq1_workstation	user_logon_req_1	User
	usrreq2_computer	user_logon_req_2	User
	usrlog2_computer	user_logon_info_2	User
DEVLEN	arl_device	app_res_list	Application
	lal_device_	logon_asn_list	User
	ch0_dev	chardev_info_0	Serial Device

	chl_dev	chardev_info_1	Serial Device
	ui0_local	use_info_0	Use
	uil_local	use_info_1	Use
DNLEN	wki0_langroup	wksta_info_0	Requester
	wkil_logon_domain	wksta_info_1	Requester
SYMBOLIC CONSTANT	COMPONENT	DATA STRUCTURE	CATEGORY
	wkil0_langroup	wksta_info_10	Requester
	wkil0_logon_domain	wksta_info_10	Requester
	usrlog1_domain	user_logon_info_1	User
DTLEN	prjob_datatype	prjob_info	Print
EVLEN	alrt_eventname	std_alert	Alert
GNLEN	grp10_name	group_info_0	Group
	grp11_name	group_info_1	Group
MAXCOMMENTSZ	grp11_comment	group_info_1	Group
	sv1_comment	server_info_1	Server
	sv2_comment	server_info_2	Server
	sv3_comment	server_info_3	Server
	sh11_remark	share_info_1	Share
	sh12_remark	share_info_2	Share
	usr11_comment	user_info_1	User
	usr12_comment	user_info_2	User
	usr110_comment	user_info_10	User
	usr111_comment	user_info_11	User
	usr12_full_name	user_info_2	User
	usr110_full_name	user_info_10	User
	usr111_full_name	user_info_11	User
	usr12_usr_comment	user_info_2	User
	usrlog2_full_name	user_logon_info_2	User
	usrlog2_usrcomment	user_logon_info_2	User
MAXDEVENTRIES (DEVLEN)	cql_devs	chardevQ_info_1	Serial Device

MAXWORKSTATIONS (CNLEN)	usr12_workstations	user_info_2	User
	usr111_workstations	user_info_11	User
NETBIOS_NAME_LEN	nb0_net_name	netbios_info_0	NetBIOS
	nb1_net_name	netbios_info_1	NetBIOS
NNLEN	ai2_netname	alias_info_2	Alias
	ae_ct_netname	ae_connstart	Auditing
	ae_cp_netname	ae_connstart	Auditing
	ae_cr_netname	ae_connrej	Auditing
	cq0_dev	chardevQ_info_0	Serial Device
	cq1_dev	chardevQ_info_1	Serial Device
PATHLEN	acc0_resource_name	access_info_0	Access
	acc1_resource_name	access_info_1	Access
	ae_ra_resname	ae_resaccess	Auditing
	ae_rr_resname	ae_resaccessrej	Auditing
	ae_cf_resname	ae_closefile	Auditing
	ae_am_resname	ae_aclmode	Auditing
SYMBOLIC CONSTANT	COMPONENT	DATA STRUCTURE	CATEGORY
	ae_um_resname	ae_uasmode	Auditing
	ae_al_resname	ae_acclim	Auditing
	fil_pathname	file_info_1	File
	fi3_pathname	file_info_3	File
	wki0_root	wksta_info_0	Requester
	wki1_root	wksta_info_1	Requester
	sv2_userpath	server_info_2	Server
	sv3_userpath	server_info_3	Server
	shi2_path	share_info_2	Share
	usr11_home_dir	user_info_1	User
	usr11_script_path	user_info_1	User
	usr12_home_dir	user_info_2	User

	usri2_script_path	user_info_2	User
	usrill_home_dir	user_info_11	User
	usri2_parms	user_info_2	User
	usrill_parms	user_info_1	User
	usrlog1_script_path	user_logon_info_1	User
PDLEN	prdest_name	prdest_info	Print
QNLLEN	ai2_queue	alias_info_2	Alias
	prq_name	prq_info	Print
	prjid_qname	prid_info	Print
RMLLEN	ui0_remote	use_info_0	Use
	uil_remote	use_info_1	Use
SHPWLEN	shi2_passwd	share_info_2	Share
SNLEN	ae_ss_svcname	ae_servicestat	Auditing
UNLEN	username	print_other_info	Alert
	username	user_other_info	Alert
	ae_so_username	ae_sesslogon	Auditing
	ae_sf_username	ae_sesslogoff	Auditing
	ae_sp_username	ae_sesspwerr	Auditing
	ae_ct_username	ae_connstart	Auditing
	ae_cp_username	ae_connstop	Auditing
	ae_cr_username	ae_connrej	Auditing
	ae_ra_username	ae_resaccess	Auditing
	ae_rr_username	ae_resaccessrej	Auditing
	ae_cf_username	ae_closefile	Auditing
	ae_ss_username	ae_servicestat	Auditing
	ae_am_username	ae_aclmod	Auditing
	ae_um_username	ae_uasmod	Auditing
	ae_no_username	ae_netlogon	Auditing
	ae_nd_username	ae_netlogondenied	Auditing
	ae_al_username	ae_acclim	Auditing
SYMBOLIC CONSTANT	COMPONENT	DATA STRUCTURE	CATEGORY

	conil_username	connection_info_1	Connection
	fil_username	file_info_1	File
	fi3_username	file_info_3	File
	wki0_username	wksta_info_0	Requester
	wki1_username	wksta_info_1	Requester
	wki10_username	wksta_info_10	Requester
	sesil_username	session_info_1	Session
	sesi2_username	session_info_2	Session
	sesil_username	session_info_10	Session
WRKHEUR_COUNT	wki0_wrkheuristics	wksta_info_0	Requester
	sv2_srvheuristics	server_info_2	Server
	sv3_srvheuristics	server_info_3	Server

Notices

August 1996

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "(C) (your company name) (year). All rights reserved."

(C)Copyright International Business Machines Corporation 1988, 1996. All rights reserved.

(C) Copyright Microsoft Corp. 1988, 1991.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth

in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AFP	AIX
AnyNet	FFST
FFST/2	IBM
NetView	OS/2
Presentation Manager	Workplace Shell

The following terms are trademarks of other companies:

1-2-3	Lotus Development Corporation
Apple	Apple Computer, Inc.
Borland	Borland International Corporation
Intel	Intel Corporation
Microsoft	Microsoft Corporation
Novell	Novell, Inc.
NT	Microsoft Corporation
Watcom	Watcom Corporation

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.